# ICL TECHNICAL JOURNAL

Volume 4 Issue 4
November 1985

ICL TECHNICAL JOURNAL

The ICL Technical Journal is published twice a year by Peter Peregrinus Ltd. on behalf of International Computers Ltd.



All correspondence and papers to be considered for publication should be addressed to the Editor.

1986 subscription rates: annual subscription £16.00 UK, £19.00 overseas, airmail supplement £8.00, single copy £10.00. Cheques should be made out to 'Peter Peregrinus Ltd.', and sent to Peter Peregrinus Ltd., Station House, Nightingale Road, Hitchin, Herts. SG5 1SA, UK, telephone: Hitchin (s.t.d. 0462) 53331.

The views expressed in the papers are those of the authors and do not necessarily represent ICL policy.

**Publisher**
Peter Peregrinus Ltd.
PO Box 8, Southgate House, Stevenage, Herts. SG1 1HQ, UK



Printed by A.McLay & Co. Ltd., London and Cardiff ISSN 0142–1557

ICL TECHNICAL JOURNAL

# Contents

**Volume 4 Issue 4**

# Foreword

On the 21st April 1985 it was announced that the Queen's Award for Technological Achievement had been presented to ICL in recognition of the successful innovation represented by the development of CAFS-ISP.

The problem of accessing large amounts of data in a structured manner, so as to present usable information within a reasonable time, began to be recognised as early as 1962. The CAFS system is a combined hardware and software solution to this problem. Additionally, it is especially well positioned because throughout the development there has been a clear understanding that any technological breakthroughs achieved in solving the problem must fit into the real world of existing customer data and equipment.

The CAFS-ISP product clearly represents a breakthrough worthy of the Queen's Award, and as such is one of the unique technological advances in the data-processing industry today. As a vehicle it is just now being recognised by other organisations trying to provide comparable data-access capabilities.

This issue of the *ICL Technical Journal* reviews the history of the development of CAFS from the first recognition and formalisation of the problem through to the evolution of the physical solution, and also the history of the supporting software. In addition it illustrates the use of the product in the real world, showing how this has extended from the original closely defined problem of data access to include the processing of text and other forms of information by real users to solve real problems.

*Asa W. Lanum*
Director and General Manager, ICL Applied Systems

# History of the ICL content-addressable file store (CAFS)

**J.W.S. Carmichael**
ICL Mainframe Systems, Slough, Berkshire

**Abstract**

The paper is a chronological review of the development of CAFS from an original notion through successive stages of research, development and integration to a standard ICL product. It is essentially the author's historical review included in the 1985 Report of the CAFS Group Working Party of the ICL Computer Users Association.

## 1 Introduction

The origin of CAFS as a concept was the explicit recognition that the activities of a living organisation of co-operating people are not totally ordered, so that their interactions must be regarded as an interplay of order and disorder. The use of information to maintain their organised behaviour is therefore also characterised by an element of disorder, in its fundamental sense of intrinsic unpredictability, and so, whenever a large amount of data is stored on magnetic media, some applications will require that it be accessed by search. The traditional approach to solving this problem has been to build indexes to cope with the more predictable searches. This is fine as far as it goes, but not all searches can be predicted, and if one tries to build indexes to cope with every eventuality – by, in effect, totally inverting the file – the task of managing the indexes, particularly when the data is volatile, can become intolerable.

Two very relevant quotations from Vic Maller[1] are:

> 'There are, in fact, instances where the indexes can occupy between two and four times the volume occupied by the data to which they refer – a perfectly absurd situation'. '... It should not be assumed that indexable operations cover the totality of useful functions ... Indexes are really very primitive projections of files and consequently their utility should not always be taken for granted'.

Content addressing, or associative processing, is a natural way of trying to circumvent these difficulties. Very natural, in fact, since we are all very familiar with the marvellous effects we can achieve using these techniques in retrieval from the memories in our heads. Laymen have always assumed that

computers work this way; look at the computers in *2001*, *Doctor Who* or *Blake's Seven* to see how ordinary people think they behave.

In the early 1960s there was much speculation in the academic world on the potential value of associative stores, but few suggestions regarding practicable techniques for making such a store of adequate capacity at acceptable cost. In 1962 Gordon Scarrott, working with Roy Mitchell in the Ferranti computer department, was stimulated by this situation to point out to their management that if a store, of adequate capacity to be useful, conceptually rotates as a consequence of its operating principle – for example a magnetic drum or a delay line – then the mean time required for access by search is half the revolution time, exactly the same as for access to a known address. Hence associative access can be achieved with existing technology, without an additional access time penalty. However, in 1962 the concept of a 'data base' had not become common and consequently the need for store access by automated search was not widely recognised. Moreover, there were no resources available for exploratory development, and so no physical work was initiated at the time.

## 2 Initial project

In 1969 George Coulouris, then lecturer in computer science at Imperial College, suggested to Gordon Scarrott a joint project with ICL's Research & Advanced Development Centre to identify the intrinsic requirements for file storage and to propose ways for meeting such requirements – a suggestion that was immediately accepted. The late Roy Mitchell, then a senior member of the staff of RADC, took responsibility for the project, collaborating with John Evans, one of George Coulouris's senior students.

At that time moving-head disc storage devices were not in common use. Nevertheless, in December 1969 Roy Mitchell had proposed a combination of disc store, key store and comparator, search evaluation unit and retrieval unit. He coined the term 'CAFS' to refer to the system, a name by which this combination of mechanisms is still known. Initially there was some confusion about whether the 'A' stood for 'addressable' or 'addressed', but this issue has long been resolved in favour of 'addressable' since the system permits addressing by content but does not enforce it.

It is noteworthy that most of the individual events (i.e. comparisons) in the CAFS searching system take place as soon as the data becomes available from the disc store, so that Roy Mitchell's design was in effect a 'data flow' system as now defined. But of course the term 'data flow' was not used in 1969.

Since CAFS is an engineering innovation, its most basic feature, the autonomous searching mechanism, should be regarded as a novel and advantageous synthesis of ends and means. Gordon Scarrott and Roy Mitchell proposed the means in 1962, George Coulouris and John Evans

recognised the ends in 1969, and Roy Mitchell proposed the synthesis in 1969. On this foundation others, notably Ed Babb in connection with hardware and Tom Addis with software, proposed additional valuable features during the development, and Vic Maller maintained the momentum of the project over a difficult period when ICL was preoccupied with defining and launching the 2900.

During 1970 proposals for investigating and evaluating the characteristics of a CAFS device progressively took shape. Coulouris and Evans undertook technical research in the USA, leading to the publication of CAFS Report No. 1: 'Some characteristics of real time data management systems'. The specification of a joint project between ICL and Imperial College was submitted for possible support to the Advanced Computer Technology Project. In June CAFS Report No. 2 gave 'An outline of the CAFS programming interface'.

In 1971 Roy Mitchell's design for an experimental machine was adopted in a project, funded under an ACTP contract, for the construction of a prototype at RADC. The principles of the machine which became known as CAFS Mark 1 were published at IFIP 71 in a paper by Coulouris, Evans and Mitchell entitled 'An approach to content-addressing in data bases'. During 1971 Vic Maller joined Roy Mitchell at Stevenage, to undertake applications studies, software management and experimental evaluation. A large number of potential applications within ICL itself were identified, including systems in Group Purchasing, Spares Inventory, Personnel, and Production Scheduling. On a wider scale, major possibilities were identified, even at this early stage, in police systems, in the hospital service, in matching jobs and applicants, in services provided by libraries and information publishers, and in the preparation of government and trade statistical analyses.

## 3 Prototypes

1972 saw the completion of the Mark 1 prototype, and the start of an extensive testing programme. It was soon confirmed that data stored on discs could be searched and retrieved by the unique hardware developed by the RADC team at speeds far in excess of those possible using traditional software techniques. The Mark 1 machine used an 8 Mbyte exchangeable disc attached to a 1900 series host, and already had the characteristic separation of functions between key store and comparator unit, search evaluation unit and retrieval unit. The Journal of the British Computer Society, *Computer Journal*, published another paper by Coulouris, Evans, and Mitchell entitled 'Towards content-addressing in data bases'[2]. The success of these trials justified initial contacts with the Post Office, for a study of the Directory Enquiries problem, and with the IEE's INSPEC service as an example of a document storage and retrieval application.

By 1973 enough experience had been acquired for the design of a replacement machine to be started. This was intended to be a further research and

experimental device, but already considerable debate had started within the company on not only how, but also whether, CAFS could eventually be incorporated into the official company product line, and what procedures and standards should be followed to transfer information smoothly between the research and manufacturing processes. Meanwhile, the Mark 1 machine was successfully demonstrated to the Post Office Directory Enquiries management, and the outline of a possible PODQ trial was agreed.

1974 saw the construction of two prototype Mark 2 CAFS machines. Although it was clear that the major future market would be associated with what at that time was called the 'new range', subsequently the 2900 series, it was natural that the CAFS developments should be continued on the stable foundation of 1900 architecture. The Mark 2 CAFS was designed to search data held on EDS 60 discs, using multiple read heads and amplifiers to read and search up to 10 tracks at a time; this multiplicity, which has sometimes been mistaken for an essential characteristic of a CAFS device, was in fact adopted as a means of boosting the data transfer rate to one which matched that of the CAFS searching and evaluation mechanism. A modified 7503 was used as the control processor, which interfaced with the host and directed the activity of the underlying CAFS components. A VDU was installed at Hemel Hempstead for preliminary experiments on Directory Enquiries; not surprisingly, these highlighted many requirements for enhancements and corrections to the pilot software, but also generated sufficient enthusiasm to justify progress to an extended trial.

In 1975 a CAFS Exploitation Review Committee was established to coordinate future policy. It considered how to take advantage of the Department of Industry's procedures for assistance under the preproduction orders scheme. Some interest was aroused in the CCA (predecessor of the present CCTA) for the possible application of CAFS in public service applications in, for example, PRISM, DHSS and the Home Office. An initial submission was also made to the European Commission. A PODQ Project Team was established within ICL.
Throughout the mid 1970s the CAFS team at RADC made steady progress in the development of software products and techniques, exploring possible ways of exploiting CAFS as a search engine more fully. Major advances were registered by Ed Babb and Len Crockford in the definition of data models to describe an online database in a manner which permitted relational processing with optimal efficiency. The facilities of the enquiry language also underwent continual pragmatic refinement, with particular attention to psychological factors affecting the usability of the enquiry commands. The implications of CAFS for database design and structure provoked much lively debate between proponents of IDMS on the one hand and of CAFS on the other; there was not much prevision of the eventual mutually beneficial synthesis of the two approaches.

During 1976 and 1977, as such development work proceeded, a programme for the transition of CAFS to full product status was developed. The

prototype applications were to be consolidated, and practical experience in live application conditions acquired at a limited number of sites, actively cooperating with ICL and with each other. This would initially require the manufacture of a small batch of machines, following the architecture of the Mark 2 engines but conforming more closely to normal manufacturing conventions. A CAFS marketing manager was appointed; formal disclosure procedures were set up to control the flow of information from the centre of the company through approved units of the sales organisations, to candidate customer pilot sites. The PODQ trial also advanced, with the transfer of the CAFS enquiry software from Stevenage to the PODQ Project Team at Bracknell, where it was modified to incorporate the various specific requirements that the pioneering experiments had defined.

During 1978 the main Post Office Directory Enquiry trials took place. Terminals in the telephone exchanges at Leeds and Leatherhead were connected to the CAFS machine installed at ICL Bracknell, where the records of 7 000 000 subscribers were held; this represented about half of the national directory. All the technical criteria for the trial were triumphantly met; for example, an average response time of 2.5 s had been specified, and over the whole trial the observed average was 1.7. There was also noteworthy and very positive feedback from the operators of the trial service. They very quickly found the best ways to use the facilities – how much or how little search information to enter, which terms were the most successful search keys, how best to proceed with incomplete information, and so forth. They also all reported that the system was much more enjoyable and satisfying to operate than the previous 'conventional' techniques. All these reactions are now commonplace, but at the time they were valuable confirmation that we had been proceeding on the right lines.

#### 4 Official product

The main event of 1979, from the CAFS point of view, was the announcement of CAFS 800 – the first marketed version of the CAFS Mark 2 machine – as an official ICL product. One of the first of these was installed in ICL's own Group Information Systems, where the first application with live data was the personnel system. Over the next two years a total of 14 such machines were installed, in very different organisations and covering an exceedingly wide range of applications. At mid 1985, 12 of these machines were still in successful active service. These machines naturally attracted a great deal of interest, and were seen as fully confirming the claims for the principles of CAFS that had been made over the years. On the other hand, with a complicated architecture and somewhat outmoded technology, they were adjudged to be expensive at some £250 000 each. Nevertheless, although it was difficult to justify such expenditure in advance, nearly all users reported very satisfactory, although hard to quantify, benefits as a result of using a CAFS service, whether in reduced tactical and *ad hoc* programming, improved quality of data, or greater accessibility of information.

The success of the CAFS 800 programme led naturally to the development and implementation of CAFS-ISP (information search processor), bringing CAFS into the mainstream of ICL's product line. From now on CAFS was to be progressively incorporated in the standard environment of VME, to become automatically associated with standard file and database structures, and to be available in the standard hardware of the 2966 family of systems. CAFS-ISP was officially launched at a well attended open meeting of the CAFS User Group – perhaps it should then have been called the CAFS Intending User Group – in April 1982.

Before the end of June 1985 the 500th order had been received for CAFS modules with 2966 family systems; of these some 450 had been installed, on over 300 systems, in a dozen countries around the world. In addition, April 1985 saw the launch of the Series 39 mainframes, with the announcement that from now on CAFS would be an automatic constituent of every ICL mainframe system.

The success of CAFS stems from the combined efforts of many people, too many for all to be cited here. So let this review conclude by signalling the invaluable contribution of two groups:

- the pioneers in RADC, without whose flair, vision and pertinacity there would have been no CAFS product
- the CAFS User Group, whose serious commitment has convinced the world of the importance of the concept and the product

**References**


1 MALLER, V.A.J.: 'The content addressable file store — CAFS', *ICL Tech. J.*, 1979, **1** (3), 265–279, (includes a useful bibliography).

2 COULOURIS, G.F., EVANS, J.M. and MITCHELL, R.W.: 'Towards content addressing in data bases', *Computer J.*, 1972, **15** (2), 95–98.

ICL Computer Users Association (UK), CAFS SIG: 'Exploiting CAFS-ISP', Working Party Report, July 1984. (2nd Amended Reprint, July 1985), ICLCUA (UK), PO Box 42, Bracknell, Berks, RG12 2LQ, UK.

# History of the CAFS relational software

**A.T.F. Hutt**
ICL Management Support Business Centre, Reading, Berkshire

**Abstract**

The paper is a chronological review of the development of the relational software that now forms part of the CAFS system. It is essentially the author's personal recollections of this history, in which he was deeply involved.

## 1 The beginning

The origins of the product go back to 1969, when the Computer Department of the BBC Television Service, under Mr C. Lashmar, built the first release of the Television Management System[1]. The purpose of this system was to maintain records of the costs incurred while producing television programmes and to provide details of these costs to those involved in the work. Once this system was in service, user reaction led to the production of several subsequent versions and to the point in mid-1972 when further developments were dependent on the use of better concepts and more advanced methods. Once this situation was recognised, Mr Lashmar approached ICL and Southampton University with a proposal to sponsor, jointly, a bursary aimed at finding a practical solution to these problems. The bursary was established in 1973 by BBC Television, ICL and the university to investigate ways of using database techniques to produce large information systems; the author of this paper was awarded the bursary and started work on the project in May 1973. So one could say that the development of the CAFS relational software started with an advertisement for a senior designer to work on this bursary for three years.

## 2 The bursary

This bursary had the advantages for all parties involved that it offered the holder and Southampton University the opportunity to perform research in a comparatively untouched field of study, and if successful would provide the BBC with a solution to problems it needed to solve and ICL with a contribution to its product line. It says much for the foresight of Professor Barron of Southampton and Mr T. Brooks of ICL that most of these hopes have been fulfilled.

At the time of starting, the overall plan for the project was to spend 6 months with the BBC reviewing their system and the rest of the time at Southampton: 12 months on system architecture and design, 12 months on system

construction and the remaining 6 months on writing up a doctoral thesis. This overall plan governed the work until August 1976 when the thesis was presented for examination. The following paragraphs describe briefly the work carried out in these periods.

### *2.1 The BBC*

The six months spent here were very instructive. I had spent the previous seven years designing the Edinburgh Multi-Access System EMAS[2] and the file store and file-management systems for VME-B[3,4,5]; consequently, this was the first opportunity I had had to investigate a large application suite. My overall impression of the Television Management System was that it was very large and it was difficult for anybody to grasp its full extent; furthermore, that instead of writing Cobol code large parts of the system could be generated using modern database-management systems.

Being familiar with both top-down system design methods and database design methods I spent a lot of effort producing a model of the Television Management System which was both comprehensive and easy to understand. Later, this proved to be the sort of system description which ICL subsequently modelled in the Data Dictionary System[6].

### *2.2 The architecture phase*

This started with a series of system studies and discussions with those working on other database systems; it was to have a major impact on the rest of the project.

At this time the main specifications for both the Codasyl and Relational Database Systems[7-11] had been produced and I was faced with understanding how they could be exploited to solve the BBC problem. Fortunately, help was at hand. In 1974 IFIP established a Working Group known as TC2, concerned with the development of database-management systems. This group was significant because it held meetings[12-14] that took a very broad view of the subject and in particular was prepared to discuss the advantages of the Codasyl and Relational systems. I joined the group and spent very worthwhile weeks in Corsica, Belgium and the Black Forest with Ted Codd, Chris Date, Peter Stocker and many others learning about their work on relational systems, and this convinced me that the solution to the BBC problem must be based on the relational model.

As a result of these inputs the specification of a system began to emerge; the main features it would support were

- an application design methodology
- a five-level schema supporting the methodology, each level taking the form of a relational database
- a family of relational languages to access both the users' data and the definitions of this data.

This system was known as the Relational Data Base Management System (RDBMS)[15-17] and is the direct forerunner of the CAFS software delivered by ICL.

### *2.3 System development*

The specification of the system having been produced, the next task was to produce the system itself; this dominated the rest of the project.

Southampton University and the BBC both had ICL 1900 computers, and after a lot of discussion it was decided, on grounds of portability, to program RDBMS in the language Coral 66. The early work was carried out at Southampton but it soon became apparent that larger computing resources were needed and the project was transferred to the 1906A at the Atlas Laboratory at Chilton in Berkshire. With a large and powerful machine, and the Atlas Laboratory staff who were extremely supportive, work progressed rapidly and in December 1975 I had an early prototype which did in fact support a large number of the system facilities. In fact, this was only the second working relational system in Britain, the only other being the Peterlee Research Test Vehicle[18], and was one of only about six world-wide, other prototypes running at the same time being System R from IBM[19] and INGRES[20].

### *2.4 Writing up*

The need to write my doctoral thesis meant that I could no longer carry on programming RDBMS; however, the BBC, ICL and the university, and by this time the National Research & Development Council (NRDC) had recognised the value of the work and put together a two-man team to continue the work.

Writing the thesis gave me the opportunity to revise and upgrade most of the specification and to carry out some performance and sizing work using the system. The thesis was presented in September 1976 and led immediately to the next phase.

## 3 Exploitation of the system by ICL

I returned to ICL in 1976 and began to consider this question: how was this novel system, the result of a successful research project, to be integrated into the ICL product line and developed so as to become a standard product available to all customers? The rest of the paper is about the solution to the problem.

### *3.1 Keeping it going*

The first task, clearly, was to ensure that the project continued. Letters to the Managing Director, setting out the situation, resulted in the work being given top management support and the project moved from its base in the Product

Marketing Group to the Product Development Group. The BBC agreed to house the project and Southampton University and the NRDC agreed jointly to the continuing use of the 1906A at the Atlas Laboratory; and all parties agreed to a continuing project staff of three.

These arrangements continued for a further 18 months until Christmas 1978, when the project was finally absorbed into ICL.

### *3.2 Decision to produce RDBMS*

The big question asked in ICL was how could RDBMS fit into the product line. In 1977 ICL was developing IDMS and the Data Dictionary System as the mainline products and was committed to the Codasyl Network Data Model; and these major products, aimed at very-large-scale applications such as very big stores management problems, provided facilities beyond the resources of RDBMS. Eventually the decision was made to produce RDBMS as a small, easy-to-use system aimed at the user who had little or no database experience but wished to develop his own simple system.

This decision led to questions about the usability of RDBMS: it would need to be changed so as to make it simpler to understand and easier to use. Thus there followed a series of workshops and reviews involving staff at all levels in the company from Ed Mack, then Director of Product Development, downwards. In the end everyone became convinced that it would be possible to turn RDBMS into a product for the market envisaged and the project was formally transferred from its research status to a product-line development.

### *3.3 The Personal Data System*

The process of simplification and redesign was extremely painful. Large parts of the research work had to be discarded because they were aimed at a higher level of complexity than was currently required, and care had to be taken that nothing was lost that might be required if and when the product was redeveloped as a full relational database system. The outcome of this work was the specification of the Personal Data System, PDS[21].

The first release, PDS 50, was for the ME29 and 2904 machines in 1980; it was later released for the 2900 VME machines as PDS 70. This was a significant milestone on the development route of the RDBMS software because it recorded that as a result of a great deal of effort in a variety of directions RDBMS had been successfully transformed from a research vehicle to a product-line item.

### *3.4 Querymaster 50*

The understanding gained from the development of the Personal Data System enabled the RDBMS project to tackle its next major challenge, the development of Querymaster.

During the 1970s it had been stated on several occasions that it would be possible to produce a relational database interface to a Codasyl system, but no such working system was produced during that decade. The first realisation of the claim was in 1981 with Querymaster 50[22] on the ME29 range, which does in fact give a fully integrated interface between IDMS and the relational system.

The development of the first version of Querymaster had the objective of providing a read-only version of the Personal Data System which would interface with traditional files and IDMS databases; furthermore, as most of the IDMS database definitions were in the Data Dictionary, the product had to interface with the latter also in order to use those definitions. Major improvements were made to this first version, as described in the next section.

There are only two other relational interfaces to IDMS; both run on the IBM version, which were produced by a Polish team led by Dr. Staniszkis[23].

### *3.5 Improving Querymaster 50: the Level 100 releases*

The version of PDS and Querymaster just described were very early versions and implement only a small part of the RDBMS functionality specified in my doctoral thesis. Experience of customers' reactions to these products led to the decision to implement the remainder: this represented a massive upgrading of the products and involved first dismantling them, then adding the new functionality and finally rebuilding them.

This major project was carried out in the first 6 months of 1982 and resulted in the first working system becoming available on field trial on the 14th July 1982 – a date well remembered by all those involved. Over a longer period the work resulted in Querymaster 110 on VME machines and 120 on ME29; and in PDS 110 on VME and 120 on ME29.

### *3.6 Relational interface to CAFS*

One of the main shortcomings of the RDBMS products was that, being aimed essentially at the end user, they provided no means for the applications programmer to use the relational facilities. This was removed by the development and release of the Relational CAFS Interface, RCI[24]; released in 1984, it provides much of the functionality of Querymaster in a form that allows an application program to process the information retrieved.

### *3.7 CAFS-ISP*

The release of Querymaster and the Personal Data System meant that the concept of end users interacting directly with the data in their machines had been firmly established, and there was now to be expected a demand for improved performance in these products.

At the ICL Stevenage laboratory the CAFS project team had been developing their hardware and software and there had been plenty of spirited discussion between the CAFS and RDBMS teams on the relative merits of different approaches. The time had now come for a joint development to meet the demands on the product line. There was vigorous debate on the virtue and possibility of marrying the CAFS machine and the relational software, and it became clear that this was the way forward. A hectic 6 months followed, during which designs and plans were put in place so that the CAFS hardware could meet the new requirements of handling ICL standard discs and data formats, and the various software products such as IDMS and RDBMS could be adapted to accommodate the CAFS hardware. The successful completion of all this work was signalled in 1983 by the release of the Series 200 Querymaster products, CAFS-ISP and all the software needed to support these.

## 4 Summary, conclusion and acknowledgments

The programme of work that has been described started 13 years ago as an academic research project sponsored by the BBC and Southampton University, and has led to a range of products that are widely and increasingly used by ICL customers throughout the world. Perhaps the most important conclusion is the very gratifying one, that the belief of those who initiated and gave early support to the work has proved so well founded.

A large number of people throughout ICL have worked to bring the programme to its present successful state; I have mentioned a few in the course of the paper and would like now to thank all who have contributed to the achievements.

## References

1 LASHMAR, C.W.: 'The control of MIS: management and methods', in 'Management information systems', Infotech State of the Art Report 21, Infotech Information Ltd., UK, 1974.
2 '*EMAS users guide*', Edinburgh University, 1970.
3 'VME introduction', ICL Technical Publication RP0071, 1981.
4 HUTT, A.T.F.: 'A data base approach to system architecture', Information Processing 1974, Proc. IFIP Congress, 5th–10th Aug. 1974, North Holland, Sweden.
5 'VME file store management', ICL Technical Publication R00183/02, 1984.
6 'Data Dictionary System', ICL Technical Publication 6514, 1982.
7 'CODASYL DBTG subset specifications', Oct 1973, available from ACM or British Computer Society.
8 CODD, E.F.: A relational model for large shared data banks', *Comm. ACM*, 1970, **13**, 377–387.
9 CODD, E.F.: 'A data sub-language based on relational calculus', Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, available from ACM.
10 CODD, E.F.: 'Normalised data base structure: a brief tutorial', Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, available from ACM.
11 CODD, E.F.: 'Further normalisation of the data base relational model', Courant Computer Science Symposium 6, 1972, *Data base systems*, RUSTIN, R. (ed.), Prentice-Hall.
12 'Proc IFIP TC-2 working conference on data base management systems', Cargese, Corsica, 1st–5th April 1974, North Holland, 1974.

13 'Proc. IFIP-TC2 special working conference on 'A technical in-depth evaluation of the DDL', Namur, Belgium, 13th–17th Jan 1975.
14 'Proc. IFIP-TC2 working conference on modelling in data base management systems', Freudenstadt, West Germany, 5th–9th Jan 1976.
15 HUTT, A.T.F.: '*RDBMS reference manual*', Southampton University, 1975.
16 HUTT, A.T.F.: 'A relational data base management system', Doctoral Thesis, Southampton University, 1976.
17 HUTT, A.T.F.: '*A relational data base management system*', Wiley, 1979.
18 TODD, S.J.P.: 'PRTV: a technical overview', IBM Scientific Centre Report UKSC 0075, May 1975.
19 ASTRAHAN, M.M.: 'System R: relational approach to data base management', *ACM Trans. on Data Base Syst.*, 1976, **1**, (2).
20 '*INGRES Reference Manual*', Relational Technology Inc., Berkeley, California, USA, 1982.
21 'Personal data system CPD57', ICL Technical Publication RP1077, 1980.
22 'Using Querymaster', ICL Technical Publication RP1143, 1981.
23 STANISZKIS, W. *et al.*: 'NDMS prototype user's guide', Rapporto CRAI 84-23, CRAI, Rende, Calabria, Italy, Oct. 1983.
24 'Relational CAFS interface user guide (RCI). 100', ICL Technical Publication R00251, 1984.

# The CAFS system today and tomorrow

**G.McC. Haworth**
ICL Management Support Business Centre, Reading, Berkshire

**Abstract**

The CAFS search engine is a real machine in a virtual machine world; it is the hardware component of ICL's CAFS system. The paper is an introduction and prelude to the set of papers in this volume on CAFS applications. It defines the CAFS system and its context together with the function of its hardware and software components. It examines CAFS' role in the broad context of application development and information systems; it highlights some techniques and applications which exploit the CAFS system. Finally, it concludes with some suggestions for possible further developments.

'Search out thy wit for secret policies
And we will make thee famous through the world'
Henry VI, 1:3

## 1 Foundations

The late 1960s saw ICL's leading architects designing the new range of computer systems that was to become the 2900 Series. Specifically, those responsible for the operating system were required to:

- provide a cost-effective environment for developing and running applications
- adopt an architecture to fulfil changing requirements and exploit new technological opportunities.

From the start, the designers acknowledged the rapid and accelerating process of innovation which characterises the IT industry. They recognised that the common computing processes would migrate from application software via system software and microcode to special-purpose hardware. The result based on the twin concepts of the virtual machine and the procedure call was the virtual machine environment, VME.

File searching is clearly a common process and was identified as such in 1969. Maintained file-projection indexes and data infrastructure are not always appropriate or cost-effective for data-retrieval. Research in this area produced the CAFS.800 search engine and development led to the CAFS-ISP

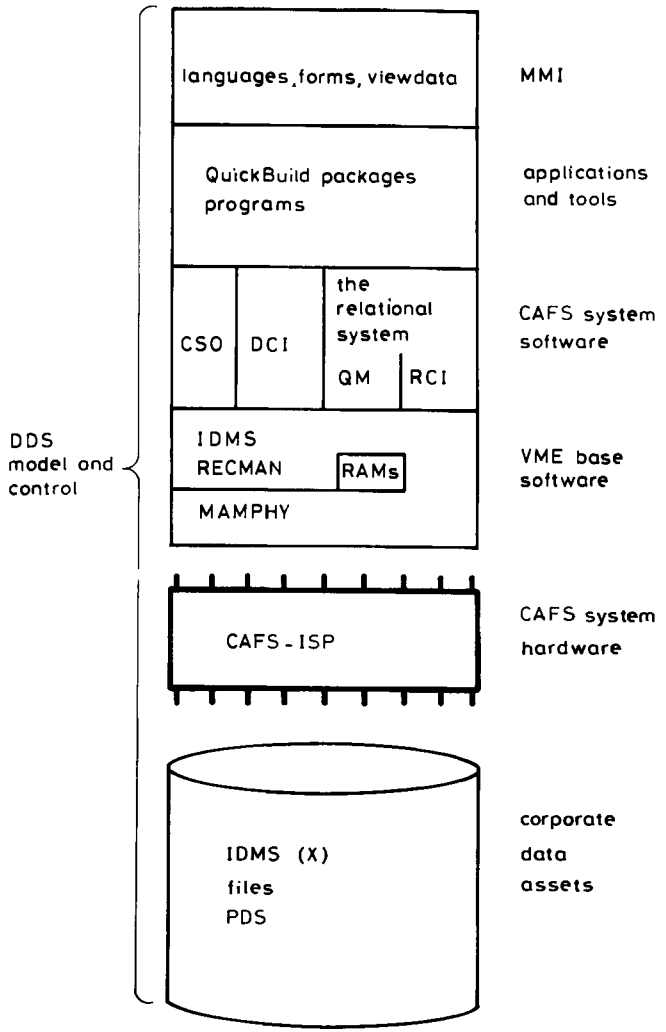


Fig. 1 The CAFS system in context

product[1]. The integration of CAFS-ISP into the VME virtual machine environment is evidence of the robust architectural foundations laid down at the outset.

## 2 The CAFS system

Initial success with CAFS.800[2] [5] demonstrated the effectiveness of the hardware approach to file searching. It achieved a quantum leap in performance, searching at 'disc-speeds' and transferring a major part of the normal mainframe load from the central processor to the peripheral CAFS engines.

ICL set out to incorporate CAFS as a standard subsystem within VME. An engine specification similar to that of CAFS.800 was combined with synergy requirements as follows:

- retrieve records satisfying criteria expressed as a combination of Boolean logic and threshold functions
- calculate during the search such derived data as are commonly required
- return a 'projected' set of hit records
- search existing data, whether stored in files or databases
- co-exist with current hardware and current workloads
- operate below existing or industry-standard interfaces

The programme resulted in what is now called the CAFS system. This is defined here and currently comprises five elements, one hardware and four software, as follows:

- the CAFS-ISP hardware search engine, here abbreviated to 'CAFS'
- the VME CAFS search option, CSO
- the direct CAFS interface, DCI
- the relational system:
  Querymaster for *ad hoc* and production data queries
  the relational CAFS interface, RCI, an extension of Cobol

Fig. 1 shows the CAFS system in its context of target data, VME support and application software.

For the sake of brevity, it is convenient to use rather a large number of abbreviations in describing the system and its functions. These are explained when first introduced and listed in the glossary at the end of the paper.

### *2.1 CAFS engine*

This is attached to a disc control module (DCM), through which data not being searched by CAFS passes directly as normally. The engine consists of five main components, as shown in Fig. 2:

- the logical format unit (LFU)
- the key channels (KCs)

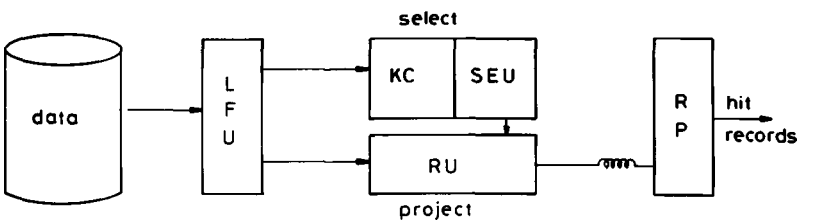


Fig. 2 The CAFS engine

- the search evaluation unit (SEU)
- the retrieval unit (RU)
- the retrieval processor (RP)

*2.1.1 Logical format unit:* The LFU provides input to the other components, advising them where to find relevant fields in the records. It has been given information about the data-file's logical block and record formats. It examines the incoming data stream, identifies starts and ends of records and fields and in some cases examines the content of the record.

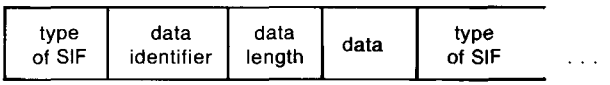


Fig. 3 The self-identifying format for data

A record that can be searched by CAFS has a fixed-length part followed perhaps by a variable-length part. The latter may be a string, an array of fixed-length fields or data in self-identifying format (SIF). The unit picks out the appropriate types of data stored in this last format, illustrated above.

*2.1.2 Key channels:* The search criterion, i.e. the set of conditions that a record must satisfy to qualify as a 'hit', is of the general form

logical condition (LC) & {interfield comparisons} (IFC)

for example:

(LC) age < 30 & 2 from [experience = insurance, banking, audit] &
(IFC) achievement > target

LC combines Boolean and quorum terms using the Boolean operators 'AND', 'OR' and 'NOT'; the terms are evaluated from the results of component atomic conditions of the form:

FIELD masked by MASK is in RELATION to LITERAL
where the relationship is $=$, $\neq$, $>$, $<$, $\geq$ or $\leq$

The example above features one Boolean and one quorum term. The Boolean term has one atomic condition; the quorum term has three.

Two features add to the CAFS functionality. First, CAFS can detect whether records satisfy nominated subconditions within LC. Secondly, CAFS can mask a field to ignore unknown or irrelevant parts of the field; this endows it with a powerful fuzzy-matching capability.

In evaluating the complete criterion, the key channels and search evaluation unit deal with LC and the back-end retrieval processor deals with IFC.

A battery of 16 key channels performs the first part of evaluating LC; not all will be needed in every case. Each channel examines one atomic condition and signals in parallel with the others and via three bit-stores whether their masked field contains value(s) less than, equal to or greater than their literal.

None of the bit-stores will be set if the field does not exist, one will be set for a single-valued item and any number will be set according to the content of a multivalued field. For example, the text item 'QUICK BROWN FOX' in SIF format when compared with FOX will use the VME EBCDIC collating sequence to signal BROWN < FOX, FOX = FOX and QUICK > FOX.

The software surrounding CAFS uses some KCs; VME for example uses one. The LC can therefore involve a maximum of 12–15 atomic conditions.

*2.1.3 Search evaluation unit:* The SEU declares whether the record satisfies LC as well as the nominated subconditions within LC. It has a battery of 16 search evaluation processors (SEPs), programmed to operate in concert to indicate in a 'task word' whether the condition LC and/or the nominated subconditions are true or false. They are assisted in this task by two further subunits as illustrated in Fig. 4. The quorum processor (QP) evaluates all, possibly weighted, quorum expressions. The select processor (SP) broadcasts the QP results and interim SEP results to the SEPs.

Finally, the SEU increments counts of records satisfying the LC condition and the nominated subconditions by examining the task word.

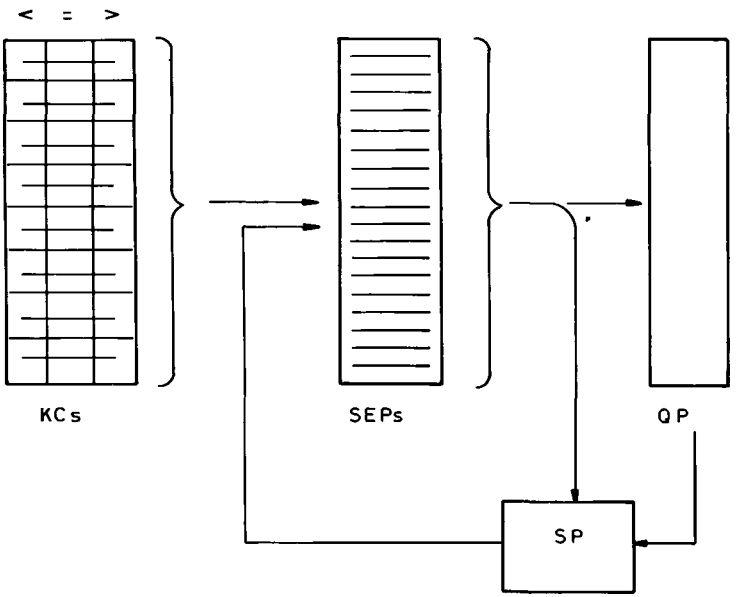


Fig. 4 The selection evaluation unit

*2.1.4 Retrieval unit.* This unit is told by the LFU which bytes of the record to retrieve as if that record had satisfied the LC. It stores retrieved records in the retrieval processor's (RP's) 32 kbyte store for analysis. The record is subsequently prefixed by an identifier, length and task word if it satisfies LC and discarded if it does not.

The RU also notes the end of a logical block of data in the RP store for checkpointing purposes.

*2.1.5 Retrieval processor:* Up to this stage, the CAFS engine works as a strictly synchronous pipeline processor. The RP's activity, however, is store-buffered as described above. This allows the front end of the CAFS engine in an extreme case to work thousands of records ahead of an RP examining a physical cluster of records satisfying LC.

The RP has two roles. One is to calculate data derived from the file search as a whole; the second is to be the final arbiter about passing data back to the host mainframe. These are separable functions; the RP, after returning a nominated number of records to the mainframe, can cease to retrieve but continue to analyse the whole file.

The RP may calculate a number of functions in a user-determined sequence. It evaluates the set of interfield comparisons constituting IFC. It computes the maxima, minima and totals of values in specified fields. At any point in this sequence, the record could be rejected as a candidate for passing back to the mainframe.

The data-delivery rate of current discs limits CAFS search speed except in high-hit-rate situations where the load on the retrieval processor can become a limiting factor. Record reduction is done by the RU in parallel with record selection but handling hit records and evaluating functions take time.

In summary, and in relational database terms, the key channel and search evaluation units jointly perform the SELECT operation and the retrieval unit performs the PROJECT operation.

It can be seen that the CAFS engine employs powerful, parallel and purpose-built hardware focused on a common task previously done by conventional von Neumann software.

The elapsed times of tasks unassisted by CAFS have been improved by factors typically of 10–100 when CAFS was introduced. At the same time, much if not almost all the work has been transferred from the central processor to the CAFS engine, as was intended; in one verified case, 99·94% of the mainframe load was removed.

Appendix 1 gives a more detailed model of the search and retrieval performance of the CAFS engine together with information showing how the

objective of coexistence has been met. The first ICLCUA report[6] includes the results of the first CAFS performance tests.

The following four sections describe the software interfaces to the CAFS engine which enable the prospective user to tap the search power of the hardware.

### *2.2 CAFS search option*

The objective of this software is to enable existing programs to use CAFS' search power without requiring any change to the code. Such programs will be ones which we do not wish to or cannot change. The former category includes operational programs which have been well run-in, low-priority 'one-offs' as well as the year-end undocumented antique of apocryphal importance and complexity. The latter category includes packages and applications generated by QuickBuild.

CSO is addressed to the classic batch suite of Cobol and Cobol-like programs which select records from a file using a known criterion and process the 'hit' records.

CSO allows selection on the basis of a Boolean-only criterion; quorum logic and interfield comparison cannot be used to eliminate records. Further, CSO cannot be used by the other three software interfaces to CAFS.

The CSO interface is a single system control language (SCL) command SET_CAFS_CRITERIA (STCC)[7] with parameters for defining the relevant record format(s) and the selection criterion. This causes selection intelligence and CAFS instructions to be interposed (Fig. 5) at record-access method (RAM) level between the target data file and the program.

For example, suppose a program ISSUEDEMAND processes a file MYMASTER containing records of several different types, each starting with a four-character field, TYPE. The program is only interested in records where:

TYPE = 4 and AMOUNT_OWING ⩾ £300 000.00

The following SCL boosts the performance of ISSUEDEMAND:

```
ASSIGN_FILE (MYMASTER, MAINFILE, LEVEL = C)
STCC (LNAME = MAINFILE,
      ITEMS = TYPE (1:4) & AMOUNT_OWING (9:9),
      CONDITION = "(TYPE EQ 4) and
      (AMOUNT_OWING GE 300000)")
ISSUEDEMAND
```

ISSUEDEMAND continues to evaluate returned records as normally

because it has not been changed in any way. It finds that each record is a hit and that the file appears to be only 0·001% or whatever of its previous size. The program completes its task a great deal sooner than before.

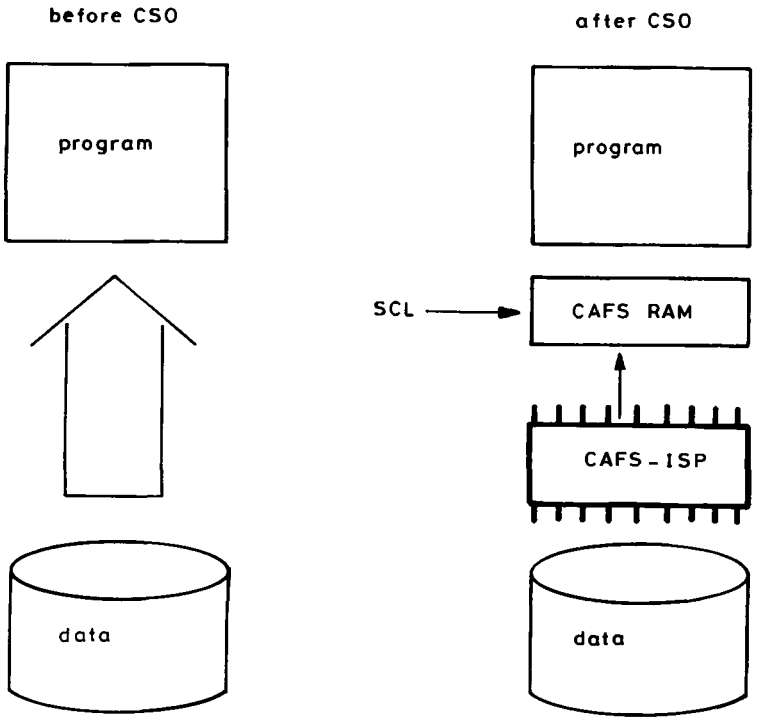


Fig. 5 Before and after CSO

The reduction in elapsed time and use of system resources is governed entirely by the hit rate on the file; the smaller the hit rate, the greater is the improvement. CSO users would normally expect to see a considerable improvement in performance, and with an interface consisting of one SCL command it is not difficult to try a number of experiments.

### *2.3 Direct CAFS interface*

The direct CAFS interface (DCI) is the only software interface with the objective of providing all the facilities of the CAFS engine to the user. It does so via the standard VME procedure call, which is almost universal and independent of the host language used. Character-string parameters in high-level syntax are passed from user-software to DCI at runtime and fully validated[8,9].

The other three software interfaces tailor CAFS' functionality to the needs of

their respective users for the sake of simplicity. For example, only DCI provides quorum evaluation, management of multiple types of SIF data, full subcondition counting and access to the full range of CAFS' RP facilities. DCI is therefore 'the closest approach to the original sound' and logistically the best way to gain hands-on experience of the CAFS engine itself.

DCI is the interface for high-level language programmers, for those not requiring the high-level relational CAFS interface and for the most sophisticated CAFS applications, possibly requiring the highest performance.

In addition to providing a procedural interface to CAFS, DCI provides facilities in the following categories:

- file management: open, read and close file
- TP transaction management: save, restore, end phase etc.
- text management: SIF to/from legible conversion, trailers
- DCI package runtime control: checks and diagnostics

The minimum sequence of DCI tasks, not using CAFS RP functions, will:

- open a file for CAFS searching
- define the target record's format and retrieval requirements
- define the search criterion LC, IFC being null
- read the next retrieved record
- close the file being CAFS searched.

Three criteria using three subconditions indicate the range of DCI's selection capability:

C1 = FEATURE EQ 'dglazing'
C2 = FEATURE EQ 'garage'
C3 = FEATURE EQ 'gardens'

- Houses below £50 000 in Finetown with two of the three features:
  PRICE LT 50000 and TOWN EQ 'FINETOWN' AND QUORUM THRESHOLD 1 C1 C2 C3
- Must have garage and double-glazing: anything near Crewe or a Cheshire house not in Shambletown:
  C1 AND C2 AND (POSTCODE EQ 'CW!' OR
  (TYPE EQ 1 AND COUNTY EQ 'Cheshire' AND TOWN NE 'SHAMBLETOWN'))
- How many properties have double-glazing or garage or garden?
  Define LC = C1 AND C2 AND C3 AND FALSE to reject all records in RU
  Request counts on the three subconditions C1, C2 and C3

DCI needs CAFS to access data, in contrast to the relational system which uses both CAFS and non-CAFS modes of data access. It addresses only

record manager (RECMAN) files but it is ICL's intention to extend it to address IDMS databases.

To date, DCI has been used from Application Master, Reportmaster, Querymaster, Cobol, Fortran77, Pascal, RPG2, SCL, Algol68 and from Filetab and Rapport.

### *2.4 Relational system*

In contrast to the two CAFS interfaces so far described, ICL's relational system was not designed especially for CAFS. Its objectives and methods were established[10] in the knowledge of parallel work on data models and CAFS.800 but before the appearance of the current CAFS engine. A common emphasis on data retrieval and the ability to search existing data provided the basis for the successful integration of the relational CAFS engine and the relational software. The results obtained should encourage ICL's customers to bring the power of CAFS to their existing and developing systems.

ICL's complete relational system is provided as three separate products; these are the personal database system PDS, the end-user query language Querymaster (QM) and the relational CAFS interface (RCI), a Cobol language extension.

PDS is classified here as part of the CAFS context rather than part of the CAFS system. PDS data is searchable by CAFS but the PDS software itself does not include CAFS-search capability. QM and RCI make automatic and transparent use of CAFS as appropriate.

The objectives of ICL's relational system were:

- to widen the community of data users
- to mesh with other data management products and preferred system development methodologies
- to retain the flexibility to exploit new techniques.

The aim of providing effective access to a wider community implies that the data should be presented to users in a simple and uniform way and that the special knowledge required of the user should be reduced to a minimum. Low-level details, for example of data storage, should be of interest only to the small minority responsible for providing a relational data-access platform to the majority. Keeping detailed information in place is essential in the division of labour necessary in any organisation. Here, to provide read-access to data, it is only the senior analyst/programmers and query-service providers who work below the relational level with more intimate details of data storage.

The relational system data model presented to the users includes a tabular

data structure without user-visible navigation links between tables. The tables presented to the end-user or the Cobol programmer are the results of using the relational operators SELECT, PROJECT and JOIN. This data model conforms with Ted Codd's restated definition of a relational database[11]. It also adopts a duality principle by allowing programming uses of the relational interface to be tested out interactively.

The data model is enriched by knowledge of the inter-relationships between the entities represented by the tables of data[12,13]. It is therefore not necessary for the Querymaster user to join tables explicitly unless there is some choice or ambiguity about the inter-relationship of one table with another. This results in shorter queries and less risk of user confusion. RCI users enjoy analogous benefits.

The relational products intended for use with shared data are designed to integrate with and work off the data dictionary system DDS. Data dictionary systems are increasingly being regarded as necessary control centres modelling both the using organisation and its systems development. DDS provides documentation on the accessibility of data by groups of users and avoids the need to duplicate information on the format or storage of data. In the case of CAFS, new dictionary definitions have been introduced to document the encoding of text data in CAFS SIF format and to control CAFS searching of IDMS areas.

The relational system addresses existing data in both IDMS and RECMAN formats; here, therefore, a high-level definition of a logical database has been coupled with a Codasyl (Conference on data systems languages) definition of a physical database and the resulting system enjoys the complementary benefits of the relational and Codasyl approaches.

It is part of the marketing mythology in the industry that the Codasyl and relational approaches to data management are in conflict; in fact they address different levels of database definition, as noted above and illustrated in Fig. 6, and fit well together, as this work has shown. The theory of update logistics and semantics at the relational level is incomplete; ICL has therefore retained the Codasyl standards for update and data-integrity enforcement.

The technological flexibility of the relational system is amply demonstrated by the integration of CAFS support and by subsequent developments such as the interface to graphics facilities on a workstation.

The design of the CAFS support for the relational system had to balance the great benefits offered by CAFS against the preservation of a simple and consistent view of data wherever these came into conflict. Every function offered by the system is supported whether CAFS is available or not in order to simplify the user interface. The system maps the large variety of Cobol data types to a smaller range of relational data types, regardless of the CAFS engine's ability to handle the physical data.

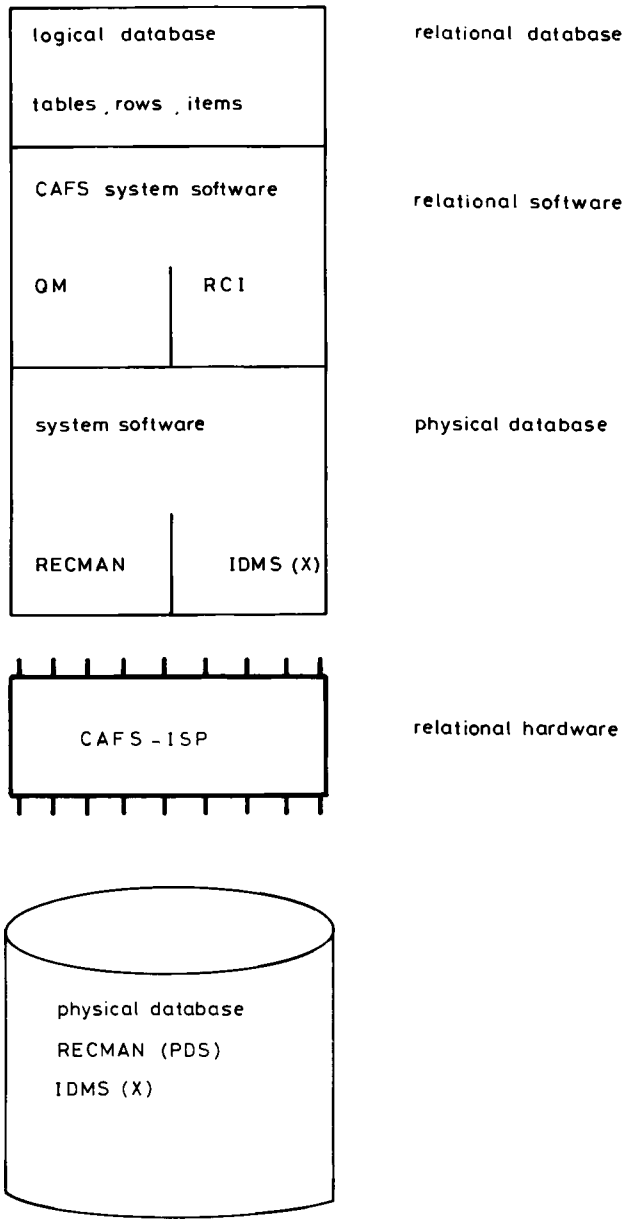


Fig. 6 Physical and logical databases

In practice, a major part of the CAFS functionality has been exploited in supporting the relational model. Comparisons are supported on all data types and the ability to handle the SIF format has been made available in its single most valuable manifestion – word-searching of free text with fuzzy

matching. The quorum capability of CAFS has not been made available directly as it does not fit easily with the implicit approach to the specification of joins. The CAFS retrieval processor is employed to count and make interfield comparisons. The combined use of (partial) primary key selection and CAFS searching is the most important performance optimisation introduced by the RECMAN CAFS software; it is faithfully exploited by the relational system.

In the IDMS context, the relational system and the IDMS database manager co-operate to perform the necessary selection and retrieval. The user's independence from the choice of RECMAN or IDMS data storage has been preserved. The introduction of CAFS support has not altered the form in which end users pose their enquiries except for the provision of word-searching and fuzzy matching.

The availability of CAFS influenced the relational system's rules for optimised data retrieval. The system looks for the best 'opening move', as most enquiries allow no choice of navigation path after selecting the first record type. This is a good approach where hit rates are small; the optimiser does not have volumetric information on which to base its choice, but users can give specific advice on the 'opening move' where this is appropriate.

The relational system on the whole uses designed-in data infrastructure such as record keys and IDMS sets and indexes. It supplements these data access techniques by using CAFS for serial or partly keyed scans whenever possible. Both access optimisation and CAFS exploitation are rule-based strategies and are defined in detail in the product manuals[14,15].

*2.4.1 Querymaster:* Querymaster provides a wide range of users with an online relational query service to shared data, typically an IDMS database with additional RECMAN files. The user conducts a dialogue with the product to select a query view; within that query view, the user can explore the availability of data and select and retrieve data by means of enquiries in a simple language. The product displays and prints tabular data, stores temporary results, sorts and provides summary information such as totals at required control breaks.

The user's task is much simplified since Querymaster selects the navigation path and resolves names to decide which record types are to be accessed, how they are to be joined and where CAFS is to be used. A query is presented as a simply structured command with data selection conditions following the keyword 'WHERE', e.g.:

```
LIST CUST-NAME, ORDER-NO, ORDER-DATE, QUANTITY,
  PRODUCT-DESC WHERE COUNTY STARTSWITH 'LANC'
  AND ORDER-DATE = 1.10.84 TO 31.10.84
```

The query view is created by the VME command CREATE_QUERY_VIEW

from a definition in a DDS data dictionary. When the shared data is fully described in DDS, the task of creating and documenting a new query view is reduced to the selection of data to be viewed and the tailoring of that view to the particular requirements of its users.

Querymaster supports the online nature of enquiry not only with CAFS but also with comprehensive 'help' facilities and parameterised macros for the significant proportion of repeatedly used 'production' queries.

The combination of Querymaster and CAFS, as illustrated by Corbin[16], enables end-users across a wide ability spectrum to express their data-retrieval requirements and satisfy their substantial latent demand for timely information.

*2.4.2 Relational CAFS interface:* RCI is a Cobol language extension, providing both a program interface to CAFS and a read-only relational interface to data. RCI looks like a serial-file handler to the programmer who is presented with relational views (qv SQL) of the data. Behind the relational interface, RCI is using the standard SELECT, PROJECT and JOIN operators to compose these relational views.

The principal objectives of RCI were to provide:

- a transparent Cobol interface to CAFS
- a read-only relational interface to IDMS/RECMAN data
- value-based privacy to add to Codasyl's item-based privacy
- additional program/data independence for simpler maintenance
- simplified programming and higher levels of productivity
- selection on and processing of text fields

The Cobol programmer manipulates the data views through four new verbs provided by the Cobol system:

| | |
|---|---|
| START | creates an instance of the view, fixing selection parameters |
| READ | delivers the next record of the view instance |
| SAVE | preserves the state of a view instance at the end of a TP phase |
| RESTORE | restores the view instance state at the start of next TP phase |

The data environment of an RCI-enhanced Cobol module includes a set of relational views known as an application view and defined in DDS. The latter is analogous to Querymaster's query view and can be tested using Querymaster.

Other features of RCI are common to Querymaster and have been covered above in the section on the relational system.

## 3 The CAFS context

Fig. 1 illustrates the wider system of which the CAFS subsystem is a part. The four elements of this are:

- the operating system: VME
- the physical data-management systems: PDS, RECMAN & IDMS
- the required man-machine interfaces (MMIs): languages, forms, Viewdata
- the system-development control mechanism: DDS

VME provides support for the CAFS system by managing resources at the record (RECMAN) and physical magnetic media (MAMPHY) levels. VME also provides job control, operator control and monitoring facilities related to CAFS.

An important synergy objective for the CAFS system was that it should address existing data on files and databases. Physical data on VME is held in personal databases (PDS), RECMAN files and IDMS/IDMSX Codasyl-standard databases.

PDS databases are in fact implemented in a published format over a set of index-sequential (ISAM) files. PDS users can deploy CAFS on PDS-held data by using DCI or, via a DDS-held retrodefinition of the files, the relational system.

The range of CAFS-searchable standard file types includes the most common ones including serial, ordered serial, index sequential and hash random but does not include files containing spanned records or nonembedded keys. The primary key of an ISAM file can be used to focus a CAFS search to a small range within the file. This facility implies that the primary key should be chosen not only to distinguish one record from another but also to provide the most useful physical record clustering within the file.

IDMS databases are made CAFS-searchable on an area-by-area basis. The relevant areas are reformatted in a single-pass process by resequencing the order of information within each page. CAFS' projection facility is used to convert physical IDMS records into subrecords as defined in the IDMS subschema; there is therefore a strong argument for tailoring subschemas to anticipated CAFS searches.

Computer systems are today being made available to a wider range of users than ever before. This has raised the relative importance of the man-machine interface in the considerations of system designers. CAFS facilities have been provided below the three key interfaces of conventional language, forms and viewdata.

Finally, it is commonly recognised today that the activities of system development need to be co-ordinated around a central model of the host

organisation and its computer systems. The data dictionary system (DDS)[17] continues to be the key ICL component for modelling and controlling all phases of such development, driving the use of all the products in the CAFS systems and adjacent to it; the preceding discussion of the relational system gives a good example of its role.

## 4 CAFS exploitation

At the moment, the number of CAFS engines ordered runs well into four figures. CAFS-capable systems can be found in all sectors of the market served by 2900 and Series 39 mainframe computers; no one sector of central or local government, public utilities, health, manufacturing, retail, insurance, education, police or defence dominates the others, nor does any single type of application dominate. There are simultaneous trends to extend the usefulness of existing systems, perform research and analysis online, create end-user services and enhance searching in operational systems – in line with the four areas of activity targetted by the four software interfaces CSO, DCI, QM and RCI. The escalation of this activity confirms the original premise that data searching is an unavoidable computing process of fundamental importance.

The application papers which follow in this issue examine specific systems which have been developed. The following notes cross-reference these applications, introduce others and highlight some useful techniques for CAFS exploitation. The second ICLCUA CAFS User Group report[18] is another useful source of application examples and techniques.

### *4.1 Using the CAFS search option*

CSO can be used in conjunction with Cobol programs and with the QuickBuild components Application Master (AM) and Reportmaster (RM) which open files in Cobol style.

It is not necessary that the selection condition employed in the Cobol program should be expressible in CSO terms. As CSO acts as a primary filter, it is only necessary that the CSO condition should be identical to or weaker than the program condition. If this is not so, use of CSO will not pass all hit records to the program and will implicitly change the role and the output of the program. Where the CSO condition is a weakened form, the program will probably reject some of the records it receives. An extreme example is where CSO is only used to retrieve records of the right type from a multirecord-type file.

### *4.2 Using the direct CAFS interface*

This section is confined to DCI-specific techniques for exploiting CAFS. It focuses on DCI's role with regard to research activity, weak typing, text, quorum searching, data profiling and software packages.

Burnard[19] and Walker[20] describe research activities in the arts and sciences which have been facilitated by DCI/CAFS. In both cases, investigations previously conducted in a batch offline stop-start mode are now being completed online. Researchers are now interacting with their data, continually testing and refining their hypotheses in a more creative environment. Clear evidence is available that CAFS is not just speeding up the logistics of research work but making it more penetrative and productive.

Research work is not of course an academic monopoly; in an increasingly competitive world, all organisations are seeking to use their resources more effectively. The ability to manage large volumes of semistructured data is a key asset in this context and is assisted by the next technique to be discussed.

The CAFS-searchable self-identifying data format (SIF) provides a latebinding mechanism whereby we can defer typing data too strongly at the data-modelling stage. For example, consider the hierarchy of object categories in Fig. 7.

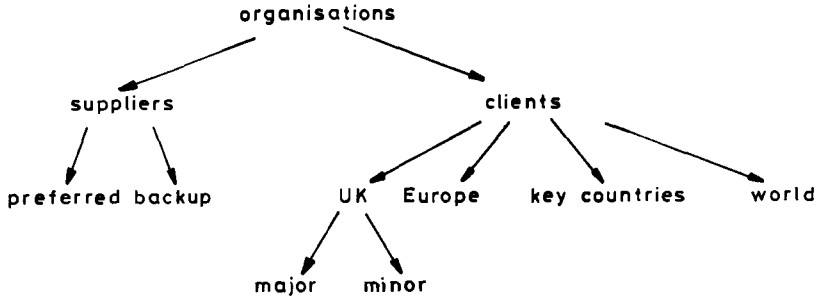


Fig. 7 A hierarchy of object categories

The categories vary from generic to specific as we move down the tree, a data model which preserves more meaning than the normal single-level entity model. Given another perspective in the model, the hierarchy becomes part of an object lattice. The data model supports queries which will vary from generic to specific; a mechanism which can type the data to suit the query in hand at runtime is clearly desirable.

SIF data, as the name implies, precedes each item of data with an identifier; data is no longer strongly typed by position. CAFS can mask the identifier just as it can mask the data values; it can ignore irrelevant or over-specific aspects of the categorisation. By a suitable binary choice of identifiers, specific categories can be converted into generic categories by CAFS masking.

Text management is one area in which varieties of 'data' will naturally occur. DCI's ability to search text is already being fully exploited and is described in detail elsewhere[18,19,21]

We examine now the uses of quorum searching. Whereas Boolean selection is

for exact data matching, quorum selection is for speculative searching. A defined but variable threshold allows records to qualify as sufficiently interesting on the basis of their proximity to some ideal target. Different factors can be weighted to reflect their relative importance. Boolean selection is appropriate when the data is determined by the selection criterion; quorum selection is better when the selection criterion is determined by the data.

For example, a personnel department will be seeking the best available candidate; a recruitment consultancy will respond with a shortlist of credible candidates; a detective is concentrating on the most likely suspects; a company naming a new product chooses a name which cannot be confused with those of its competitors' products; an inventor seeks out patents which are adjacent to the topic of his invention. Again, quorum selection is preferable to Boolean selection if the input data for the selection criterion is unreliable.

Quorum evaluation incidentally illustrates the necessity of searching; it is a calculation which cannot in general be supported by file indexes.

DCI helps the user to infer some facts from a mass of detailed data. CAFS' ability to accumulate a number of counts can be used to profile the values of a particular data item, to demonstrate the shape of the data. A distribution function can be built up by one or more CAFS scans; the results can then be displayed, say via Querymaster, in histogram, pie-chart or some other preferred form of management graphics. A subsequent interaction might focus on some sector of the distribution in search of greater detail; users might include market researchers, quality-control engineers and statisticians.

Turning now to the software industry, a number of software houses are actively working to interface their packages to CAFS via DCI. It supports fully dynamic data-management tools and has low runtime overheads. Logica with Rapport are the first to bring a CAFS version of their product to market; others are in the pipeline.

Where packages have not been developed to exploit DCI implicitly, the user may still have the opportunity to do so if the product is an 'open' one with user procedures. The QuickBuild components Application Master, Reportmaster and Querymaster all come in this category; they can call DCI modules direct as well as using RCI via Cobol. The two CAFS interfaces QM and DCI can be usefully combined together to give both a relational interface and full CAFS selection capability.

### *4.3 Using Querymaster*

Corbin's paper[16] describes the experience of one large user in setting up and using an end-user service. Although computer literacy is on the increase, the introduction of end-user computing in an organisation may involve a change of culture, attitudes, strategy and procedures.

There is a clear trend to providing higher and higher levels of management and decisionmaking with direct IT support. Data processing (DP) departments are delivering support systems to operational staff, professional IT workers and the middle if not the top ranks of management.

The availability of production and *ad hoc* query facilities is a key component of this system provision. Querymaster supported by CAFS facilitates *in situ* enquiry on core corporate data, data extraction from core data to information centres and enquiry within an information centre.

The Querymaster documentation clearly defines the 'queryview controller' role of the individual responsible for setting up one or more end-user query services. His use of such queryview-tailoring facilities as subsetting, renaming and macros can help end-users significantly.

Experience at Southern Water[16] shows its staff adding use of the query service to their regular work pattern. Where query requirements can be covered by a number of predefined macros, the site might consider integrating that query activity with existing TP services through use of RCI.

DP departments will find that good core systems attract peripheral data use of an occasional or *ad hoc* nature. The profile of data use may even change. They will also find that the provision of ready data access highlights the quality of the existing data or the lack of it; the importance of this most intangible of corporate resources will be very obvious.

### *4.4 Using the relational CAFS interface*

RCI presents serial files to a Cobol program in the same way as Querymaster presents tables of information to the end user.

RCI can therefore be used simply to boost the performance of programs which are already searching in serial fashion. This is similar in style to CSO's use except that a simple transformation of the code is required, substituting RCI calls for the existing Cobol calls.

RCI can provide single-stream input from multiple files as required by such products as Reportmaster (RM). It can select on text at word level without the Cobol program needing to recognise this new format. It can simplify the read-only manipulation of IDMS data, substituting a serial-file interface for Codasyl DML (data-manipulation language) and its implicit and sometimes difficult currency concept.

The Viewdata interface is perhaps the most attractive interface to computer systems for the general public; it has a directive screen format in familiar colour television packaging. Viewdata applications on ICL mainframes use the Bulletin TP/Cobol application product.

Berkshire County Council were the first to couple Viewdata and CAFS' search power using RCI; they have provided the public with a CAFS-based library catalogue searchable from some ten terminals in Reading's new central library, replacing constrained searching by author, subject or title by free matching on title. Berkshire demonstrated the value of the Querymaster/RCI duality; they prototyped the program's relational interface with QM and successfully ran the system three working days after taking receipt of RCI.

The deliberate similarities between QM and RCI have another benefit. As the pattern of *ad hoc* and production enquiry identifies itself, a QM service can be partially replaced by a screen-based production query service using RCI. This has usability and operational benefits in that form-filling is simpler than the QM language syntax and the TP environment is more closely controlled than the MAC environment.

RCI has raised the level of the interface between program and data, with three distinct effects. It has enabled programmers to get their programs right earlier; it has removed sources of error, replacing program-coding by a DDS-definition process, and it has increased the independence between program and data, insulating code from such changes as a migration from files to database or vice versa. It has also brought discernible productivity improvements by taking on the common chores of data manipulation which have to date been the lot of all programmers.

A major and skilled site, estimating a telephone-directory enquiry system as a 2 man-year project, completed the work in 13 man-days by combining fourth-generation QuickBuild MMI techniques with RCI/CAFS.

### *4.5 System synthesis*

A wide range of design considerations have been described elsewhere[6,18,21,22]. This section therefore confines itself to the framework for information system analysis and design.

A suitable framework will include a methodology and will support the use of a range of development tools. The development process will identify a number of phases, for example:

> IT strategy, system prioritisation, business system specification, business requirements specification, design, implementation, testing, transition, live running, maintenance.

Each phase will have defined input, output, decision points and criteria against which to optimise; most importantly, there will be criteria defining whether to proceed, double back or abandon the development process.

Given a clear framework, it should be clear where CAFS-related decisions

need to be made. Although different methodologies differ even in their naming and bounding of the above phases, some comments can be made. Note particularly that it is always possible to lose the clarity of the phased approach by making low-level decisions too early.

In the CAFS context, the business-system specification giving the high-level definition of the system's scope can be more ambitious. A broad range of medium-priority requirements will be supported without major design effort. In the business-requirements specification, CAFS will improve the feasible performance targets that can be defined. Finally, CAFS simplifies the design process by removing the need for some of the data infrastructure with consequent benefits for the remaining phases.

## 5 Future directions

The CAFS system can be developed in many ways. This section is the author's personal view of the options available.

On the hardware front, it is reasonable to assume that future implementations of the current CAFS engine will keep pace with disc and data-input technology, taking advantage of VLSI and more powerful constituent microprocessors. As the balance of system costs changes, it is possible that CAFS capability could migrate further from the system centre, from the disc control module to the disc drive itself.

CAFS microcode improvements have already come through in the product.

### *5.1 Further integration*

A likely feature of ICL's product development will be the further integration between the components of the CAFS system and between the CAFS system and its environment. Already, VME has extended the range of CAFS-searchable files and improved coexistence between CAFS and TP activity; both IDMS and Querymaster have been further tuned to facilitate CAFS searches. ICL's intention is that DCI will be supported by DDS and extended to IDMS data; in the wake of DCI, the software industry is integrating a number of packages with CAFS.

The further integration of indexing and CAFS-searching techniques is an interesting prospect, covered in detail elsewhere[21,23,24]. A 'secondary index' or 'coarse index', a search cell index rather than a record index, focuses CAFS searching onto relevant subsections of a file. This technique has already been essential on major projects with large files[24] and could usefully be made generally available as a development of VME's RECMAN facilities. Tagg[21] models the performance impact of secondary indexing.

Knowledge-engineering technology will have an increasing role to play in the future CAFS system. Pilot systems have been developed using Adviser for

performance sizing and end-user guidance. In the latter case, a dialogue with the user leads to the generation of the appropriate Querymaster command.

### *5.2 Text and office*

The CAFS engine was designed with both data and text in mind. Querymaster and RCI handle elementary text on a small scale while DCI provides more comprehensive text-manipulation facilities. CAFS-exploiting 'text' tools and applications have not yet surfaced, although Kay[23] gives an indication of the possibilities.

CAFS development will continue in the context of future international standards on character-representation and text management.

### *5.3 Relational database engine*

The current CAFS engine performs selection and projection on physical records or relational tables. Babb[25] has shown how further specialised hardware can be developed to perform these operations on logical records or general relational views. The hardware simulates joining data by performing selection on a virtual join.

International standards are crystallising at the Structured Query Language (SQL) level; SQL is likely to be more useful as a meeting point for the computer industry than as a language for computer users. The definition of standards in this and other areas encourages the development of customised hardware systems to support those interfaces.

## 6 Summary and conclusions

This paper has defined ICL's current CAFS system, has described the functionality of its hardware and software components and has shown how they work together and in the context of the VME environment. It has given some indication of the way the software interfaces are exploited, as a prelude to the other articles in this volume.

CAFS successfully performs the generic process of data searching with data-driven parallel hardware. Encouraged by the story so far, the author has hazarded an opinion of CAFS future development.

We live in an age of increasing hardware design capability. With increased customisation, we may look forward to the classic general-purpose von Neumann mainframe being replaced by open systems of specialised hardware-based servers. The CAFS system is an example of such a server.

## Acknowledgment

In my turn, I acknowledge with pleasure the significant efforts of past and

present colleagues whose combined skills have produced the CAFS system. I thank Tom Lake, now of Intercept Systems, for his first-hand insights into the architecture of ICL's relational software and the ICLCUA(UK) CAFS Working Party for stimulating discussions over the past four years. Finally, I thank Jack Howlett for his comments on this paper but claim sole credit for any remaining errors and omissions.

## References

1 CARMICHAEL, J.W.S.: 'History of the ICL content-addressable file store (CAFS)', *ICL Tech. J.*, 1985, **4** (4), 352–357.
2 SCARROTT, G.G.: 'Wind of change', *ICL Tech. J.*, 1978, **1** (1), 35–49.
3 MALLER, V.A.J.: 'The content addressable file store – CAFS', *ICL Tech. J.*, 1979, **1** (3), 265–279.
4 CARMICHAEL, J.W.S.: 'Personnel on CAFS: a case study', *ICL Tech. J.*, 1981, **2** (3), 244–252.
5 CROCKFORD, L.E.: 'Associative data management system', *ICL Tech. J.*, 1982, **3** (1), 82–96.
6 ICL Computer Users Association (UK) CAFS SIG: 'Exploiting CAFS-ISP', Working Party Report, July 1984 (2nd Amended Reprint, July 1985), ICLCUA (UK), PO Box 42, Bracknell, Berks. RG12 2LQ, UK.
7 'VME programmer's guide', ICL Technical Publication R00475/01, 1985.
8 'Direct CAFS Interface programming guide (DCI.100)', ICL Technical Publication R00421/01, 1985.
9 'Direct CAFS Interface reference card (DCI.100)', ICL Technical Publication R00431/01, 1985.
10 HUTT, A.T.F.: 'History of the CAFS relational software', *ICL Tech. J.*, 1985, **4** (4), 358–364.
11 CODD, E.F.: 'Relational database: a practical foundation for productivity' (1981 ACM Turing Award Lecture), *CACM*, 1982, **25** (2), Feb.
12 CODD, E.F.: 'Extending the database relational model to capture more meaning', *ACM Trans. Database Syst.*, 1979, **4** (4), 397–434.
13 BABB, E.: 'Joined normal form: a storage encoding for relational databases', *ACM Trans. Database Syst.*, 1982, **7** (4), 588–614.
14 'Using DDS to prepare a query view (QM.250)', ICL Technical Publication R00434/01, 1985.
15 'The Relational CAFS Interface: user guide (RCI.100)', ICL Technical Publication R00251/01, 1985.
16 CORBIN, C.E.H.: 'Creating an end-user CAFS service', *ICL Tech. J.*, 1985, **4** (4), 441–454.
17 'Data Dictionary System: the DDS model (DDS.700)', ICL Technical Publication R00408/01, 1985.
18 ICL Computer Users Association (UK) CAFS SIG: 'CAFS in action', Nov 1985, ICLCUA (UK), PO Box 42, Bracknell, Berks. RG12 2LQ, UK.
19 BURNARD, L.: 'CAFS and text: the view from academia', *ICL Tech. J.*, 1985, **4** (4), 468–482.
20 WALKER, D.: 'Secrets of the sky: the IRAS data at Queen Mary College', *ICL Tech. J.*, 1985, **4** (4), 483–488.
21 TAGG, R.M.: 'CAFS-ISP: issues for the application designer', *ICL Tech. J.*, 1985, **4** (4), 402–418.
22 'CAFS exploitation – a practical guide', ICL Technical Publication R30053/01, 1985.
23 KAY, M.H.: 'Textmaster – a document-retrieval system using CAFS-ISP', *ICL Tech. J.*, 1985, **4** (4), 455–467.
24 WILES, P.R.: 'Using secondary indexes for large CAFS databases', *ICL Tech. J.*, 1985, **4** (4), 419–440.
25 BABB, E.: 'CAFS file-correlation unit', *ICL Tech. J.*, 1985, **4** (4), 489–503.

## Bibliography


1 'IDMS part 2: database establishment (IDMS.400/IDMSX.400)', ICL Technical Publication R00154/03 (3rd Amended Reprint), 1985.
2 'IDMS part 3: using a database (IDMS.400/IDMSX.400)', ICL Technical Publication R00155/03 (3rd Amended Reprint), 1985.
3 'IDMS part 4: database programming (IDMS.400/IDMSX.400)', ICL Technical Publication R00156/03 (3rd Amended Reprint), 1985.
4 'IDMS part 5: database design (IDMS.400/IDMSX.400)', ICL Technical Publication R00153/03 (3rd Amended Reprint), 1985.
5 'Using Querymaster (QM.250)', ICL Technical Publication R00433/01, 1985.
6 'Running Querymaster in VME (QM.250)', ICL Technical Publication R00435/01, 1985.
7 'Querymaster (QM.250) user's reference card', ICL Technical Publication R00436/01, 1985.


## Appendix 1

*Connectivity, coexistence and performance*

The trend has been to increase mainframe to disc channel connectivity. This increases the accessibility of the data and reduces the performance-interference between separate processes. Greater mainframe-CAFS connectivity also implies that a major search task can be syndicated to a larger battery of CAFS engines working in concert; ten engines can search about 25 Mbytes/s.

The CAFS engine attaches to the DCM on 2966-family machines (2953, 2957, 2958, 2966 and 2988) and on Series 39. Single-OCP 2900s can connect to six CAFS engines; dual and superdual configurations can connect to eight engines. On Series 39, the connectivity is greater. Single-node Level 30s can connect to 36 CAFS engines and Level 80s to 72.

The Series 39 DCM is the high-speed disc controller (HSDC) and connects to one CAFS engine. On the 2900, the DCU/2 disc-control unit and more commonly the decision support controller (DSC) unit connect to two CAFS-compatable DCMs.

The maximum disc-drive strings on the various controllers are:

- HSDC: 8*FDS 300 or 4*FDS 2500 or 16 MDSS 'retained' drives
  MDSS drives are EDS 80s, FDS 160s or FDS 640s
- DCU/2 DCM: 16 MDSS drives
- DSC DCM: 32 MDSS drives

We have already seen that some of the synergy and coexistence objectives are being met. CAFS attaches to standard DCMs working with standard discs. It is also the case that CAFS searches most standard RECMAN files, PDS databases and IDMS databases. In the case of IDMS, a single-pass reformat procedure sets up areas of the database for CAFS searching.

Coexistence objectives also require that a 'long' CAFS search should be

interruptible by a 'short' transaction processing task. It would probably be undesirable for a single-record fetch to queue behind a 40-track full-cylinder CAFS scan on an FDS 640. VME therefore fragments CAFS searches, each fragment searching consecutive blocks on a disc-cylinder and being no longer than a system-parameter defined number of tracks.

On 2966s etc., the maximum-fragment parameter default of ten tracks can be changed at system setup time to any value. The value must be chosen to balance the needs of the existing workload against the need to exploit CAFS searching.

On Series 39, multiblock fetches and CAFS searches travel second class; single block fetches travel first class. The HSDC exercises the right to pre-empt long tasks on the disc channel when a short task arrives. Given this degree of HSDC intelligence, the maximum-fragment parameter is unnecessary.

CAFS can search data at some 3·6 Mbytes/s, outrunning the delivery rate of the fastest FDS 2500 drives. In practice, therefore, the following parameters always affect the data search speed of a CAFS engine:

- DS: the maximum formatted-data delivery rate of the disc drive
- BF: the blocking factor; block-size choice effect on delivery rate
- FF: the fragment factor; governed by the temporal dissection of the CAFS search into search fragments
- PF: packing factor; the proportion of the data blocks occupied by the records relevant to the CAFS search.

The basic upper limit on searching speed is therefore:

$$\text{file search rate} \leqslant \text{DS*BF*FF Mbytes/s}$$
$$\text{data search rate} \leqslant \text{DS*BF*FF*PF Mbytes/s}$$

Other factors such as disc head movement, rotational latency, file fragmentation, buffer management and process multiplexing all subtract from the data rate as perceived by the application program or the end user. Note, however, that these aspects of performances are standard and preceded the introduction of CAFS.

The disc speeds DS of the drives are 1·17696 Mbytes/s (MDSS), 2·22910 Mbytes/s (FDS 300) and 2·83277 Mbytes/s (FDS 2500).

The blocking factor BF is also specific to the drive concerned. Below are listed BFs for the three drives where the block size is chosen as $N$ kbytes or as a maximal value for that number of blocks/track:

Blocking factors are more significant on the faster drives and the best block sizes vary from drive to drive.

**MDSS disc drives**

| Block size, bytes | Blocks/ track | Bytes/ track | Blocking factor |
|---|---|---|---|
| 2048 | 9 | 18 432 | 0·939641 |
| 3072 | 6 | 18 432 | 0·939641 |
| 4096 | 4 | 16 384 | 0·835237 |
| 6144 | 3 | 18 432 | 0·939641 |
| 9216 | 2 | 18 432 | 0·939641* |
| 18 432 | 1 | 18 432 | 0·939641 |
| 2057 | 9 | 18 513 | 0·943770 |
| 3155 | 6 | 18 930 | 0·965029 |
| 4801 | 4 | 19 204 | 0·978997 |
| 6447 | 3 | 19 341 | 0·985981 |
| 9739 | 2 | 19 478 | 0·992965 |
| 19 616 | 1 | 19 616 | 1·000000 |

**FDS 300 disc drives**

| Block size, bytes | Blocks/ track | Bytes/ track | Blocking factor |
|---|---|---|---|
| 2048 | 15 | 30 720 | 0·804525 |
| 3072 | 10 | 30 720 | 0·804525 |
| 4096 | 8 | 32 768 | 0·858160 |
| 6144 | 5 | 30 720 | 0·804525 |
| 9216 | 4 | 36 864 | 0·965431* |
| 18 432 | 2 | 36 864 | 0·965431 |
| 2100 | 15 | 31 500 | 0·824953 |
| 3412 | 10 | 34 120 | 0·893568 |
| 4404 | 8 | 35 232 | 0·922690 |
| 7316 | 5 | 36 580 | 0·957993 |
| 9300 | 4 | 37 200 | 0·974230 |
| 19 092 | 2 | 38 184 | 1·000000 |

**FDS 2500 disc drives**

| Block size, bytes, | Blocks/ track | Bytes/ track | Blocking factor |
|---|---|---|---|
| 2048 | 18 | 36 864 | 0·785142 |
| 3072 | 13 | 39 936 | 0·850570 |
| 4096 | 10 | 40 960 | 0·872380 |
| 6144 | 7 | 43 008 | 0·915999* |
| 9216 | 4 | 36 864 | 0·785142 |
| 18 432 | 2 | 36 864 | 0·785142 |
| 2164 | 18 | 38 952 | 0·829613 |
| 3188 | 13 | 41 444 | 0·882688 |
| 4276 | 10 | 42 760 | 0·910717 |
| 6356 | 7 | 44 492 | 0·947606 |
| 11 476 | 4 | 45 904 | 0·977679 |
| 23 476 | 2 | 46 952 | 1·000000 |

The fragment factor (FF) reflects the fact that the CAFS search is fragmented by VME. The interfragment overhead is typically two disc revolutions or some 33 ms:

$$FF = (\text{max fragment size})/(\text{max fragment size} + 2)$$

Taking into account the parameters DS, BF and FF and adopting the asterisked block sizes, we can calculate 'typical' effective CAFS file search speeds as 0·922 Mbytes/s (MDSS), 1·793 Mbytes/s (FDS 300) and 2·290 Mbytes/s (FDS 2500).

The last parameter to be discussed is the data packing factor PF. Data is not usually 100% packed in a file for many reasons. Varieties of red-tape accompany the object data, several record types may coexist in a file and dynamic data should be packed at lower densities to avoid overflow.

Low packing densities have a proportionate effect on the data search rate of CAFS, but this does not mean that they subtract from the value of CAFS. Unused file space and non-target data types are the first examples of CAFS' effectiveness in filtering out disc space which is irrelevant to the search and to any subsequent processing.

The Series 39 figures below indicate retrieval processor times for various tasks. In the worst case of short records and a high hit rate, the RP component of CAFS will not keep pace with the search rate at the CAFS front end.

130 μs = overhead per hit record (260 μs on 2966s etc.)
$(L+8)/3{\cdot}5$ μs = output transfer time for the $L$-byte record
87 μs = overhead per function call
200 μs = function adding totalling on an 8-byte field
250 μs = function comparing two 2-byte fields
500 μs = function comparing two 20-byte fields

When the high hit rate is a local phenomenon, a cluster of records satisfying the LC condition, the buffering within CAFS helps to maintain the output performance of the engine.

**Appendix 2**

*Glossary of abbreviations*

| | |
|---|---|
| ADRAM | alien data record access method |
| AM | Application Master |
| AV | application view (RCI) |
| AVM | application virtual machine |
| BF | blocking factor |
| CAFS | content-addressable file store |
| Codasyl | Conference on data systems languages |

| | |
|---|---|
| CSO | VME CAFS search option |
| DCI | direct CAFS interface |
| DCM | disc control module |
| DCU/2 | disc control unit |
| DDS | data dictionary system |
| DML | data manipulation language (Codasyl, IDMS) |
| DS | disc speed |
| DSC | decision support controller |
| EDS | exchangeable-disc store |
| FDS | fixed-disc store |
| FF | fragment factor |
| HSDC | high-speed disc controller |
| ICLCUA | ICL Computer Users' Association |
| IDMS | integrated data management system |
| IFC | interfield comparisons |
| ISAM | index sequential access method |
| ISP | information search processor |
| KC | key channel |
| LC | logical condition |
| LFU | logical format unit |
| MAMPHY | physical magnetic media |
| MIP | misleading index of performance |
| MMI | man-machine interface |
| OCP | order code processor |
| PDS | personal database system |
| PF | packing factor |
| PLI | programming language instruction |
| QM | Querymaster |
| QP | quorum processor |
| RAM | record access method |
| RCI | relational CAFS interface |
| RECMAN | record manager |
| RM | Reportmaster |
| RP | retrieval processor |
| RSI | restricted system interface |
| RU | retrieval unit |
| RV | relational view (RCI) |
| SCL | system control language |
| SEP | search evaluation processor |
| SEU | search evaluation unit |
| SIF | self-identifying format |
| SP | select processor |
| SQL | Structured Query Language |
| STCC | SET-CAFS-CRITERIA (SCL for CSO) |
| SV | system version (software set) |
| TNF | third normal form |
| TPMS | transaction processing management system |
| VME | virtual machine environment |

# Development of the CAFS-ISP controller product for Series 29 and 39 systems

**N. Macphail**
ICL Mainframe Systems Division, West Gorton, Manchester

**Abstract**

The paper describes the development of the standard product-line CAFS-ISP. It covers objectives, design, implementation and enhancement capability.

## 1 Overall objectives

The corporate strategy for CAFS was agreed by Christmas 1979. A product would be developed for the 2900 series based on the CAFS 800[1,2] but with facilities more suited to a standard integrated product. Extensions of the strategy would result in CAFS being used on other systems as VLSI technology matured.

The words 'standard integrated product' meant that:

- CAFS-ISP should search existing customer disc drives instead of specially modified disc drives.
- The existing customer data should be searchable without modification.
- The product should interface to VME in a way consistent with existing practice, adding the selection and retrieval facilities of CAFS without detracting from normal direct-access disc work.
- The existing superstructure products such as Querymaster, PDS, IDMS etc. should be able to utilise the new facilities using existing methods.
- The technology used should be state of the art to enable the CAFS-ISP to be fitted as standard by reducing the cost and size substantially.
- The availability of CAFS-ISP as a standard product would allow software products to be planned to capitalise on its speed of operation.
- The design should make allowance for the next generation of disc drives.

## 2 System design

Over the next year systems resolution meetings were held to specify the top level design. As a result four specifications were derived (see Fig. 1):

- The Relative Interface specification PSD which defined the interface between superstructure products and the VME operating system. This

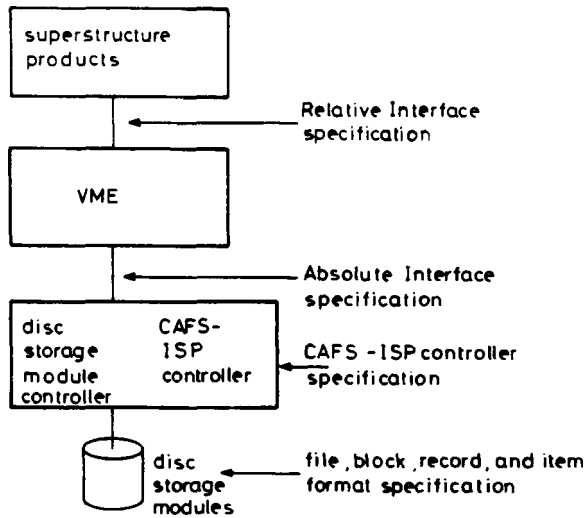


Fig. 1 Structure of system interfaces and related specifications

PSD is now superseded by the Common Target Machine – CAFS PSD 2.8.32.

- The Absolute Interface specification PSD 42.9.8 which defined the interface between the VME operating system and the CAFS controller.
- The CAFS controller specification PSD 44.4.1 which defined the functionality of the hardware.
- The file, block, record and item formats searched by CAFS-R specification PSD 46.33.1.1, which defined the data structures CAFS would search.

It was this structure of specifications which enabled product design of the various systems components to proceed independently of each other at Bracknell, Reading, Kidsgrove and Manchester. The product plans were similarly co-ordinated and controlled.

### 3 Realisation of objectives

Mainframe Systems already had a Device Control Unit on 2900 series, known as DCU2, which utilised a Universal Fixed and Exchangeable Disc Storage coupler UFEDS2. The UFEDS2 connects to EDS 80, EDS 160 and EDS 640 disc storage modules. The design approach was to connect to these drives, which transferred data at a maximum of 1·2 Mbytes/s. To allow for future devices the design target was set at 3·6 Mbytes/s. So that data could be continually retrieved by the search engine, simultaneously with post processing by a microprocessor, the retrieval buffer for retrieved data was designed to operate at 7·2 Mbytes/s, allowing alternate beat operation. The concept of

multiplexing disc streams was abandoned because the data rates of the discs were fast approaching the capabilities of the integrated circuit logic then in use (see Fig. 2).

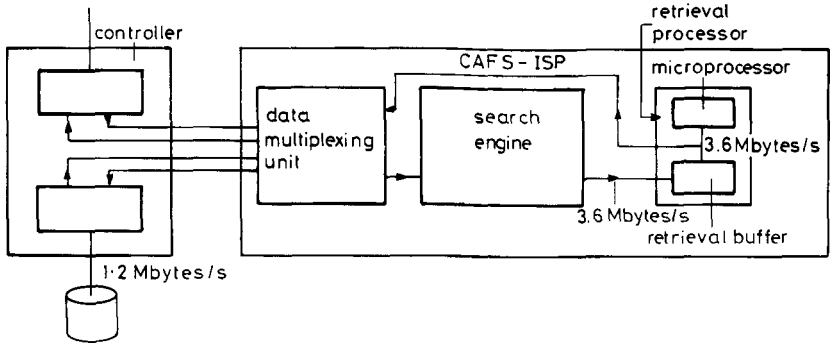


Fig. 2 Overall block diagram

CAFS 800 used a special data format, CAFS Self Identifying Format (SIF). This format was especially useful for text as it allowed, in effect, multiple variable-length fields. However, standard 2900 formats were based on one or more fixed fields followed by an optional single variable field. A new method of working was defined, called Table Defined Format, which allowed CAFS-ISP to search VME formats. To accommodate Self Identifying Format this was made a special case of the variable field in Table Defined Formats.

The structure of CAFS was now significantly different from CAFS 800. No longer was there a back-end processor with its own dedicated specially modified drives. CAFS-ISP used the existing drives with a converted standard disc coupler and standard system software. The advantages of this standard approach have been manifold.

## 4 Technology

CAFS 800 was implemented in small-scale-integration (SSI) on 6000-series printed circuit boards (PCBs). There were approximately 76 integrated circuits per PCB and over 100 PCBs. The whole was contained in a full-size cabinet.

The state-of-the-art technology in 1980 was medium-scale technology, transistor-transistor logic (TTL), on 8000-series PCBs with 30 PCBs per logic bin, two logic bins per cabinet. The DCU2 used one logic bin offering two UFEDS2 disc couplers. We had to fit two CAFS into 30 PCB positions. Furthermore, as we were using converted UFEDS2 couplers, we would have to fit these into the new logic bin. These considerations reduced us to only nine PCBs per CAFS: a reduction factor of over six to one in integrated

circuits. We could now fit two CAFS-ISP units together with their disc controllers in half a cabinet.

We chose low-power Schottky TTL to keep the compact design cool. We designed in medium-scale integration and proprietary very-large-scale integration (VLSI) and used programmable logic arrays where possible to reduce the design size. Many a system resolution meeting centred around whether one function was superior to another, each trying to occupy the last few integrated circuit positions while the design team fought to reserve room for modifications. Well, we made it.

## 5 Design structure

The design was conceived as comprising six units (see Figs. 2 and 3):

- the data multiplexing units, which, by being inserted in the dataflow, allowed CAFS-ISP to scan disc or mainframe data as required
- the logical formatter unit (LFU), whose purpose is to unravel the data structures, signalling starts and ends of records and fields, directing key channels to examine relevant fields and marking fields and logical block headers for retrieval

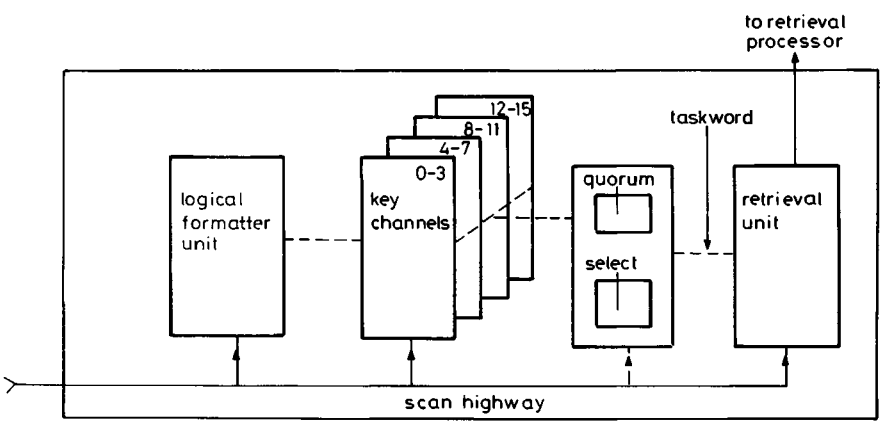


Fig. 3 Block diagram of search engine

Key:
——— data -------- control

- the key channel units which contained the keys being searched for as well as their bit masks for fuzzy searching. Another component of the key channel PCBs were the search evaluation processors (SEPs) containing the Boolean expressions which determined both the relative and logical operators combining the results of masked key comparisons of the data

stream. 16 key channels and SEPs are accommodated on four PCBs
- the search evaluation unit now consisted of only the select processor and the quorum processor. The former enabled selective feedback of SEP results into other SEPs. The latter enabled the various Boolean terms to be assigned weights and for decisions as to whether records were hits to be based on majority logic by testing for a user-defined threshold being exceeded. The outputs of the SEPs are reported in a taskword to the retrieval unit. The topmost bit marks the record as a hit or a miss
- the retrieval unit counts the incidence of the taskword bits other than the most significant bit for list count purposes. The CAFS engine has three levels of pipelining by record. This means that a record is scanned by the key channels as it comes from disc and is stored in a retrieval buffer by the retrieval unit. During the next record the search evaluation is completed. During a third possible record the retrieval unit writes away the taskword and length of the first record into the retrieval buffer store. If the taskword is marked as a miss then the retrieval unit merely restores its pointers to the start of the first record in the retrieval buffer and proceeds to overwrite it with the second. Otherwise it writes the second record immediately following the first. Obviously the second record and third records have to be stored somewhere while the first records processing is completed. The retrieval unit contains first-in-first-out buffers to cope. The retrieval unit also contains storage for the field identifier store used in SIF searches to mark selected identifiers for retrieval
- the retrieval processor contains the 32 kbyte retrieval buffer as well as the microprocessor which performs post processing such as totalling and interfield comparisons. The microprocessor is supported by working store, program store and support logic. Resident control programs are held in programmable read-only memories. Their purpose is to carry out diagnostic functions and to support loading and dumping of the main control program. There are 16 kbytes of resident program and diagnostic test data. The loadable main control program is 32 kbytes long. Both programs are written in Pascal with CALM (assembly language) sections for fast operation where required.

The logical formatter, key channels, search evaluation and retrieval units are properly the CAFS high-speed engine logic where all operations are at the rate disc data arrives or faster. The logic is sychronous and runs at 3·6 Mbytes/s. Some of the engines beats are idle beats if no data is available! (This is the majority of time on the EDS 80 family of disc drives.)

The data multiplexing unit provides the means of interfacing the CAFS-ISP to the host UFEDS. The retrieval processor provides a protocol engine to match VME requirements as specified in the Absolute Interface specification. Both of these units can be considered as support units for the CAFS engine and are candidates for future further integration with the disc controller.

The performance of the engine springs from the 16 parallel groups of key channels and search evaluation units as well as the parallel select and quorum

processors all running at clock speed. Conventional software-based search engines are orders of magnitude slower: one only needs to consider the number of instructions required for simple key channel comparisons and masking, multiply by 16 and the data rate to form an impression of the processor power required.

The engine itself will keep pace with disc data even at 100% hit rates. However, post processing by the retrieval processor microprocessor becomes an overhead at high hit rates, degrading the engine performance. The design intent is to perform in low hit rate conditions, i.e. to find 'a noodle in a hatrack', or a needle in a haystack. It is also a design intent for the CAFS traffic to be about 10% of the total disc traffic, i.e. the new fast operations of *ad hoc* enquiries should coexist with existing processing such as TP, batch and MAC. Unlike CAFS 800, CAFS-ISP is intended for use on live data which is continuously being updated by customer applications.

## 6 Overview of VME CAFS-ISP operations

The user's enquiry is passed to VME where the task is scheduled along with other tasks. VME looks at the physical search area required by the task and splits it into a number of fragments, for example ten tracks. Fragments from different tasks are then sent to the disc controller. CAFS-ISP can only process one fragment at a time. The disc controller is initialised (the *seek*) and proceeds in parallel to move the disc heads to the correct cylinders. When a disc drive is in the right position to read the data VME sends the rest of the command sequence.

In the case of CAFS-ISP the command sequence is

- *set partial results*
- *set task specification*
- *retrieve data*
- *read partial results.*

The purpose of the *read partial results* is to save counts of taskword bits and the results of post processing functions such as totalling until the next fragment is sent for that task. It follows that VME restores the partial results just before the next fragment for the same task is searched.

Each time a fragment is sent the CAFS-ISP is provided by VME with a compiled set of search criteria for that task in the *set task specification.* The data in this command deals with items such as

- a description of the logical block header structure
- a description of the record field structure, which key channel to search which field, the lengths of the fields
- the unmasked key data for each key channel
- the mask values for each key channel

- the compiled search evaluation program for each search evaluation processor
- the quorum and select processor programs, the number of taskword bit counters in use
- various retrieval parameters such as the identifiers in SIF format which should be retrieved as well as the functions to be used in post processing.

CAFS-ISP can be provided with over 13 kbytes of such data. Normally only about 2 kbytes are sent.

The *retrieve data* command causes the disc data to be scanned and hit records which are in the retrieval buffer to be transferred to the host.

Of course, events may occur which prematurely terminate a search fragment. These events could be errors detected in the search, its data or the hardware or just the exceeding of a system or unit limit. In such circumstances CAFS-ISP must offer the ability to restart at the correct error-free or complete unit of search. CAFS-ISP and VME use the logical block to define this unit of search. CAFS always checkpoints after each logical block to achieve correct recovery. The user is of course unaware of this activity.

If a disc cyclic redundancy error (CRC error) is detected by the disc controller then VME will recover the data and correct it. In this case a modified command sequence is sent by VME. A new command *write scan data* is sent immediately before the *retrieve data* command. The effect is that data is scanned from the mainframe instead of the discs so that the now corrected data is properly included in the search. The search then continues normally on the next fragment.

## 7 The Decision Support Controller (DSC)

Following the release of the CAFS-ISP to customers a further product the DSC was implemented. The objective of this product was to further integrate the CAFS-ISP into the Device Control Unit while increasing the number of disc storage units connectable to a UFEDS controller.

The result of this development was an increase in disc connectivity from 32 to 63 disc storage modules. ICL could now offer a DCU, a UFEDS with 32 streams and a CAFS-ISP in half a cabinet instead of a whole cabinet. CAFS-ISP was now integral with the controller. By adding features a further UFEDS and CAFS-ISP could be added to give the full connectivity.

## 8 CAFS-ISP on Series 39

The challenge of CAFS-ISP on Series 39 was to couple the search engine to the new High Speed Disc Controller (HSDC). This meant being able to interface with the new C8K CMOS technology, coexist in the new low-line

cabinets and match the fast operation of the FDS 300 and FDS 2500 disc storage modules.

The changes to the design were in the areas of the data multiplexing unit and the retrieval processor. The data multiplexing unit now accommodated the increased transfer rates of the HSDC as well as the differences in interfaces. Series 39 incorporates advances in maintenance diagnostic techniques so CAFS was enhanced to carry out its own static testing and to report errors both to VME and to a customer display panel to allow remote diagnostics. The resident control program in PROM was doubled to 32 kbytes to accommodate the test programs and data.

The retrieval processor also accommodates slight changes in protocol deriving from the LAN connection philosophy on Series 39. The whole was squeezed into the low-line cabinet which in one instance contains the CAFS-ISP, two FD 300 drives, the HSDC and power supplies: a further increase in integration of units. The HSDC itself integrated the DCU as well as the UFEDS, and now we have drives as well, in the same cabinet.

A further implementation of CAFS-ISP was for the FD 2500 controller where dual CAFS-ISPs were implemented, allowing dual-channel CAFS operation with the associated HSDCs.

It has been mentioned that CAFS contains a lot of memory, 13 kbytes in the search engine and 96 kbytes in the retrieval processor. The former is one of the deciding factors in selecting a VLSI technology. The close coupling of logic and memory in units operating in parallel was not achievable effectively in gate array designs such as C8K. Even state-of-the-art gate arrays containing memory are not sufficiently flexible. The future for CAFS lies in the VLSI technologies of standard cell or full custom.

## 9 Future enhancements

The exploitation of CAFS-ISP in ICL products will continue along the lines set by the corporate strategy and Statements of Direction.

- New disc storage modules with increased capacity and faster transfer rates will require a faster CAFS engine.
- The need to reduce cost will bring about further integration and size reduction using VLSI techniques. The VLSI CAFS-ISP should be more suitable for integration with products which are lower in the price range and more populous than mainframe products.
- New facilities to search text and further relational operations such as joins and projections are likely: see the paper by Babb on the file-correlation unit[3].

The benefits of fast selection and retrieval will continue to permeate existing products and applications while engendering new applications for our

customers. It is already clear, and is illustrated by the papers in this issue of the *ICL Technical Journal*, that the CAFS concepts are of fundamental importance in data processing and that the field of application of the device is continually expanding.

## References


1 CARMICHAEL, J.W.S.: 'History of the ICL content-addressable file store (CAFS)', *ICL Tech. J.*, 1985, **4** (4), 352–357.

2 HAWORTH, G.McC.: 'The CAFS system today and tomorrow', *ICL Tech. J.*, 1985, **4** (4), 365–392.

3 BABB, E.: 'CAFS file-correlation unit', *ICL Tech. J.*, 1985, **4** (4), 489–503.

# CAFS-ISP: issues for the application designer

**R.M. Tagg**
Independent Consultant, 6 Beechwood Avenue, Little Chalfont, Bucks. HP6 6PH

**Abstract**

CAFS-ISP brings new possibilities in improved response time and throughput on VME applications. The paper suggests some approaches for designers who have to decide the optimum exploitation of CAFS-ISP in proposed applications, often in conjunction with other performance tuning techniques.

## 1 Introduction

In spite of the best efforts of its progenitors and its marketing, CAFS has become regarded in many users' minds as some sort of performance panacea which will somehow obviate much of the effort in designing physical databases.

In the author's opinion, such a view is both unjustified and ill advised. A better view is that a new tool has been added to the designer's toolbox. Having an extra tool does not necessarily make the design task simpler, unless that tool renders several other tools obsolete. However, the balance of when to use each tool will change, and there may need to be a rethink by the designer before confidently using the fuller set of tools to obtain optimum performance in all cases.

This paper explores some of the design issues raised by the arrival of the CAFS-ISP tool in the designer's toolbox, and suggests some approaches which could be applied in some of the typical applications where CAFS could be used in conjunction with other performance tuning tools.

## 2 The application designer's environment

### *2.1 Available software*

A designer with the task of delivering a VME application within tight performance requirements is becoming increasingly likely to have the CAFS-ISP option available. Historical background and technical definition of CAFS-ISP are contained in the companion papers by Hamish Carmichael[1], and Guy Haworth[2], respectively.

As well as such facilities as RCI, DCI and CSO*, which are geared to the development task only, the designer may also have such tools as ISAM, IDMS/IDMSX and Querymaster, all of which could form building blocks within the operational system finally offered to the users. In some cases more specialised facilities such as those based on the ADMS database system (see Reference 3) may be used. Press announcements have also been made about the potential use of certain third-party relational DBMS such as Rapport or SIR in conjunction with CAFS. Finally secondary indexing facilities can be introduced either through IDMSX, through a third-party DBMS or through specially tailored extensions to the files in question.

### *2.2 Classes of applications*

Many approaches have been taken to classifying applications. For the purposes of this paper and for helping to indicate contrasting situations in designing with CAFS, the following six-dimensional breakdown is proposed:

*Dimension 1: Input/processing mode*

- *Batch*, i.e. waiting, either for logical or performance reasons, until a specified amount of data has arrived before proceeding with further processing
- *TP*, i.e. carrying out processing in immediate consequence of each individual unit of arriving data

*Dimension 2: Output mode*

- *Data driven*, i.e. produced as part of the processing which is automatically triggered by batch or TP data input
- *User driven*, i.e. produced as a result of a request by a user for an *ad hoc* query, or a refinement of a previous query, or by invocation of a 'canned' report procedure

*Dimension 3: Volatility*

- This is a spectrum between those applications where data arrives at a high rate and those where the database is relatively static. TP mode usually implies high volatility, but batch could be either high or low

*Dimension 4: Data sharing*

- This is again a spectrum. Some applications involve data which is accessed by many users, by many different functions or access paths, or even by other applications beyond the designer's immediate concern. Others are stand-alone or function-specific

*These and other abbreviations used in the paper are interpreted in the Glossary forming Appendix 2 to the paper by Haworth[2] in this issue.

*Dimension 5: Data structure*

- Also spectrum-like: at one extreme is the case of a single main entity-type of a relatively high number of occurrences, with a number of subsidiary 'look-up' entity-types. At the other extreme there might be a number of equally major entity-types with various relationships between them, as well as minor entity-types. The simpler structures could possibly be forced into a single file or record-type; the latter could not

*Dimension 6: Data value format*

- *Strictly formatted data*, i.e. predefined attributes with corresponding data values. Each record (entity) occurrence is expected to have a value for every attribute
- *Sparsely formatted data*, i.e. where attributes and their values are only recorded where they occur
- *Free-format data*, i.e. where no structure other than that implicit in interword spaces and other punctuation is apparent. Free text would fall in this class, but text as a value of a specific attribute would imply one of the first two classes
- *Non-character-encoded data*, i.e. image, digitised voice etc.

In practice, the designer's application will often involve a number of subapplications, each with its particular blend in these six dimensions. An initial question which must be asked is which of the subapplications are the more fertile grounds for potential use of CAFS? Once the answers to this question have been determined, the use of CAFS must then be optimised in connection with other tools. The next section of this paper discusses the initial question, after which the specific optimisation issues are addressed.

## 3 Identifying the fertile subapplications for using CAFS

Adopting a very broad brush, one could characterise a suitable CAFS-ISP subapplication as:

- any input mode
- output mode is user-driven
- volatility is low to medium
- data sharing is in terms of multiple *ad hoc* questioners of the *same* data view (rather than involving fundamentally different views)
- data structure is simple or of limited complexity (in the terms used in the previous section)
- any character-encoded data value format

Justifications for the comments on each dimension are offered below:

*Input/processing mode:* CAFS is a means of accelerating output rather than input, so it is irrelevant whether input is by batch or TP.

*Output mode:* Data-driven output, especially when triggered by TP input and processing, is largely selected by the use of single identifiers or keys which specify one particular point in the database to be accessed. CAFS does not accelerate such single-key searches.

Some automatic reports triggered by batch processing may, however, be accelerated by CAFS, although it is possible that the turn-round cycle of the batch processing may not justify the faster timings produced by CAFS. But again, machine capacity constraints would be eased by farming out of the searching effort to the CAFS unit.

With user-driven output, the advantages of CAFS are best seen on, first, invocation of 'canned' reports; and secondly, on 'longstop' searches. A 'canned' report is one whose selection criteria and format are already defined, but parameters may be supplied at run time, and each invocation operates on the state of the data in the files at a particular time. The user requires the report 'regardless', that is to say he does not need to carry out preliminary research queries to find out whether the selection criteria are sensible.

If the user wishes to test the water before initiating potentially long (even with CAFS, on very large files) searches, he will probably be assisted more by secondary indexes (which could provide quick counts of likely hits without accessing the data) than by CAFS. However unless all data is indexed, which is expensive on both storage and updating, there is always a risk that indexes may not be able to help give any hit volume estimates. This is referred to as the 'longstop' case – the user has no option but to embark on the file scan, and here CAFS will make a significant difference.

*Volatility:* High volatility can detract from the speed advantages of CAFS by progressively upsetting the match between the logical and physical sequence of a file. In normal ISAM terms, this would be manifested through increasing overflow which would periodically have to be reset by a reorganisation which might be expensive and inconvenient on a large active-round-the-clock file. A tailored 'change file' approach might be more efficient, but still has a similar problem in terms of periodic consolidation of changes into the main file.

There is also the question of whether enquirers require up-to-the-minute (or even second) accuracy, and, if so, what about updates to the end of the file that arrive after the search has already started? The tradeoffs in this area will be discussed in more detail later.

*Data sharing:* Different users may require fundamentally different views involving the same data – for example one user department might be interested in customer orders and another in the finished product stocks in warehouses. This may require a more normalised database design which could act against the interests of efficient CAFS searching, which depends on a single record type with all data relevant to each occurrence contained in one record of the file.

*Data structure:* By the definition in Section 2 above, the complex structures are the ones where it is difficult to force the data into a single record type (sharing or no sharing), hence bringing the same difficulty as for the previous dimension. One example would be a police incident reporting system where it was required to keep separate details on suspect persons and on suspect vehicles and to allow these to be related in many ways.

*Data value format:* Strictly formatted data presents no problems, nor does sparsely formatted data which is particularly appropriate to the CAFS self-identifying format (SIF). Free text can also be handled in SIF by defining a field code which precedes each word in the text, together with trailers that delimit sentences, paragraphs, sections etc. The overhead of the field codes and trailers is offset by the saving on the detail of indexing (see Sections 4.1 and 4.5 below). At present, the masking and comparison facilities, although they can operate at the bit level, are not sophisticated enough to go beyond character encoding e.g. to bit patterns of image or voice except, in theory, for exact equality.

## 4 Detailed design issues

### *4.1 Use of secondary indexing in conjunction with CAFS*

*4.1.1 Rationale for use of secondary indexes:* Although the impression is sometimes given that secondary indexes are made redundant by the application of CAFS searching, this has never been the mainstream of CAFS thinking (see Reference 4). Secondary indexing is still required if it is a user requirement to have responses to queries, based on conditions involving non-prime-key fields, within a small number of seconds on files of any size. A discussion of the use of secondary indexes in a very-large-scale project (the computerisation of PAYE) is given by Wiles[5].

Table 1 gives some examples of the limits of what can be achieved with CAFS alone when the user response time requirement is given at various levels.

**Table 1 Achievements using CAFS alone**

| Required response time | | Maximum file size assuming CAFS searching at | | Examples of files of |
|---|---|---|---|---|
| | | 1 Mbyte/s (FDS 640) | 2·8 Mbyte/s (FDS 2500) | |
| 2 s | 'immediate' | 2 Mbyte | 5·6 Mbytes | 20k–50k stock items<br>4k–10k letters/memos |
| 10 s | 'panel-game speed' | 10 Mbyte | 28 Mbytes | 10k–30k personnel records<br>1–3 filing cabinet drawers |
| 1 min | 'while you wait' | 60 Mbyte | 168 Mbytes | 300k-800k potential clients<br>100k–300k abstracts |
| 1 h | 'come back later' | 3600 Mbyte | 10 080 Mbytes | regional health index<br>insurance policy files<br>inland revenue<br>credit records |

Even though some of the longer times may be acceptable for a final answer to a query, they will often be unsuitable when the user is in 'research mode' – e.g. if he is trying to find out how many records meet a given subcriterion before searching on a more compound condition. Searching for a 'needle in a haystack' is often a multistage operation – users will not be happy if after an hour's searching the message comes back 'no hits found' or, worse, '500 000 hits found'.

So as well as for some searches which are known to require very quick response, secondary indexing is often also required to support 'count-only' queries which are preliminary to a main search – 'immediate' or 'panel-game' speed is usually wanted.

Secondary indexes are also often needed to support queries which involve relational joins between different files,since otherwise the second (and further files) will have to be searched repeatedly. This issue is addressed separately in Section 4.3 below.

Finally, there may also be a need for asecondary index for updating purposes. The ideal primary key (if ISAM is used) is one which helps to limit the range of CAFS searches on many occasions (e.g. surname in a personnel file). However, updating may be on some other key (e.g. personal number), access through which could be provided by a secondary key to allow immediate updating.

*4.1.2 Optimising the use of secondary indexes with CAFS:* Various authors[4] have proposed combining indexes and CAFS by the use of relatively coarse indexing, where an index entry points to a 'search cell' rather than an individual record. The search cell, which can vary in size between a block and a cylinder, is then subsequently searched by CAFS for the precise location of the qualifying data and any non-index-aided conditions.

This suggests that there should be a way of deciding the optimum search cell size for a particular set of circumstances. The formula below expresses that part of the search time which depends on search cell size:

$$T_1 = [1-(1-h)^{x/r}][tx + a]c/x \quad (+\text{index search time})$$

where $T_1$ = search time per cylinder of file-extent
$h$ = hit rate (proportion of *records* hit)
$x$ = search cell size (bytes)
$r$ = record length (bytes)
$t$ = CAFS search time per byte (inverse of delivery rate)
$a$ = CAFS cell search initialisation time
$c$ = cylinder size (bytes)

The assumptions on which this formula is based are:

- the query uses one criterion, supported by a secondary index
- hits are spread over the file completely at random (no fortuitous clustering

between secondary key values and primary key values)
- a binomial distribution of hits per search cell applies
- the file is large enough and the hit rate low enough that the search cell start-up time $a$ is constant (average seek + average rotational delay + CAFS unit setup).

The index search time is omitted from further analysis as it is likely to be small compared with the other part. In any case the only situation where the time would depend on $x$ would be if the index was held as a series of bit maps with one bit per search cell for each key value.

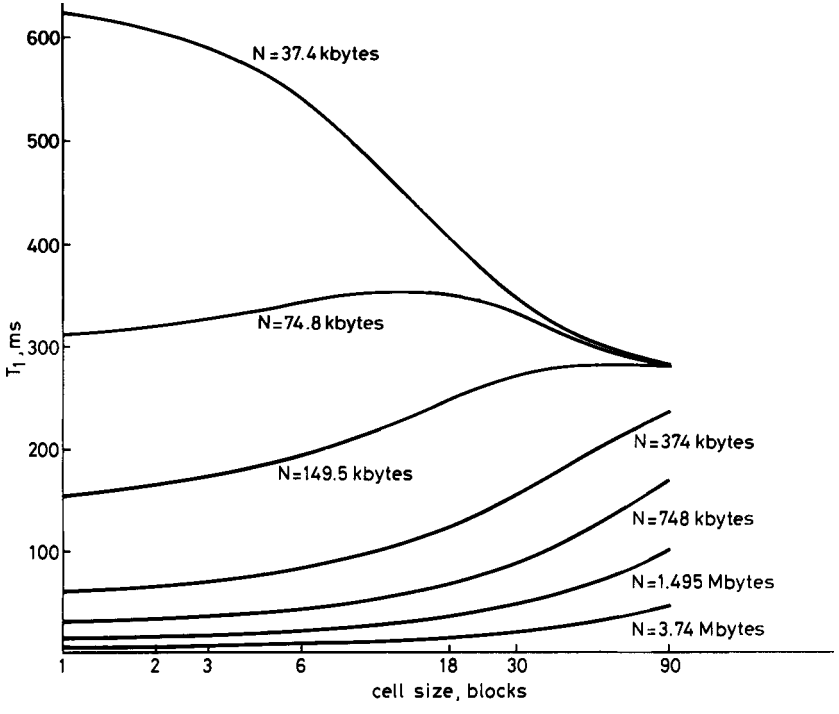


Fig. 1 Variation of search time $T_1$ with size of search cell

Cell-by-cell searches

| | |
|---|---|
| FDS 2500 | 7476 bytes per block |
| six blocks per track | hit rate 1:$N$ bytes |

The family of graphs in Fig. 1 show how the resulting value of $T_1$ varies with different choices of $x$ and $h/r$ (the hit rate per byte of file). Values for the constants were chosen as follows:

$r = 200$ bytes
$t = 1/2{\cdot}8$ $\mu$s (equivalent to FDS 2500 delivery rate of 2·8 Mbyte/s)
$a = 32$ ms
$c = 15 \times$ FDS 2500 tracks

A block size of $\frac{1}{6}$ of a track has been assumed; the search cell must be an integral number of blocks.

Perhaps surprisingly, there are no minimum points on the graphs which would indicate an optimum search cell size in between the smallest possible and the largest possible. Instead there seem to be two cases: for hit rates above around 1 per 100 kbytes of file the maximum cell size is best, while for lower hit rates the minimum is optimal. The first case approximates to full CAFS search of the whole file, whereas the second case in extreme terms implies minimum use of CAFS and relying primarily on the indexing.

The 1 per 100 kbytes dividing line for hit rate applies to FDS 2500 discs. For FDS 640 the comparable figure is around 1 per 30–40 kbytes.

Possibly the most serious criticism of the model used to establish the graphs above is that each search cell hit is assumed to require a separate initialisation time. Thus if two adjacent search cells were hits, they would both require the 32 ms overhead. An alternative model has been considered which assumes that the record-handling software interfacing with CAFS could be altered to provide some optimisation:

$T_2$ is the search time per cylinder assuming that if adjacent search cells are hit the 32 ms overhead is only required for the first cell of a string of adjacent cells in each cylinder.

The formula used is as follows:

$$T_2 = [1-(1-h)^{x/r}][tx + a(1-h)^{x/r}]c/x + a[1-(1-h)^{x/r}][1-(1-h)^{c/r}]$$

The family of graphs analogous to those for $T_1$ is shown in Fig. 2.

Fig. 2 does show a small area where minima occur, i.e. for hit rates around 1 in 74 000 bytes, but the minima are all very near the smallest cell size, certainly less than one track. It also pushes up the hit rate required to justify large cell sizes to around 1 in 40 000 bytes (around 1 in 15 000 bytes on FDS 640).

The only other assumption that might significantly affect the graphs further is the level of clustering of key values. A case in point is if the primary key of an ISAM file were 'Surname' and a secondary index existed on the Russell Soundex Code. A hit on Soundex Code would be able to be limited to specific areas of the file, since the first character is the same in both keys and the succeeding digits in the Soundex Code do represent possible groups of alphabetic letters. However, it is felt that these cases are special and that no general rules can be applied.

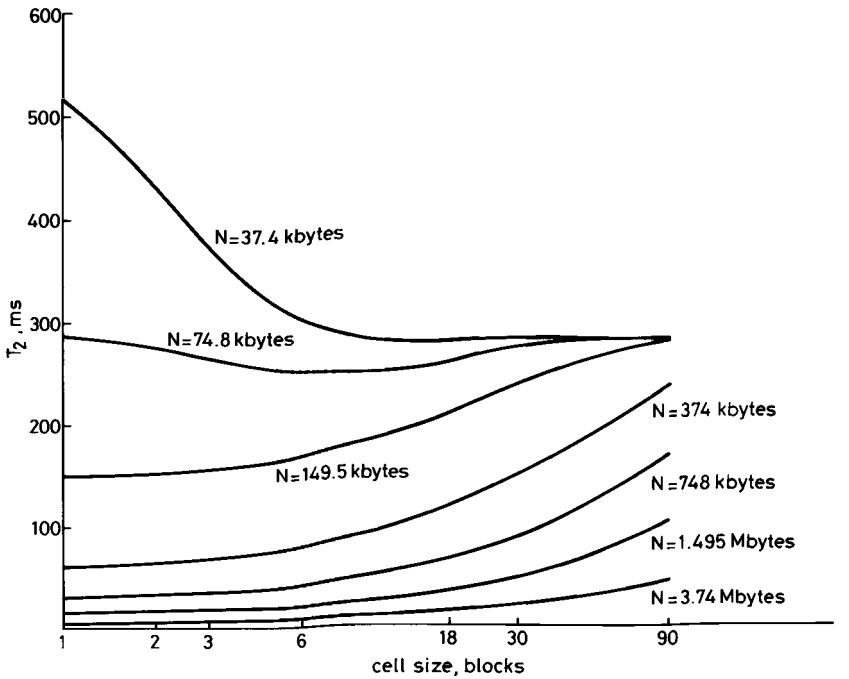


Fig. 2 Variation of search time $T_2$ with size of search cell

Cell string searches

| | |
|---|---|
| FDS 2500 | 7476 bytes per block |
| six blocks per track | hit rates 1:*N* bytes |

The conclusion appears to be that small search cells are the best unless a very high hit rate is the norm, or unless searching is restricted to a secondary key with a significant level of clustering. In practice, of course, hit rates will vary very widely. Very often a user will want estimates of the number of records hit before embarking on a retrieval. With a secondary index these can be performed entirely within the index. If the hit rate is very high, and the user requires retrieval, then it may be faster to initiate a full scan without use of the indexes. Otherwise the index/small cell combinations would be used.

### *4.2 Balancing retrieval and update performance in volatile applications*

If a retrieval subapplication is preceded by a TP or batch one, or if the timing of updating of the database is critical, then there will often be a design balance to be struck between optimising for retrieval and optimising for updating.

In theory, the single-record-type, denormalised data structures preferred by CAFS make the updating task less complex. But in practice some of the methods which are good for retrieval can degrade update performance, e.g.

- associated secondary indexes
- maintaining a match between physical sequence and logical primary key sequence
- holding multiple-valued fields
- duplicating identifiers and other data from 'parent' entities

*Secondary indexes*, as we have seen in Section 4.1, may have to be used in conjunction with CAFS. Updating secondary indexes immediately for each update transaction is very time-consuming if volatility is high. Deferred index updating is often practised (transactions are batched for the index update run) with the new data searchable sequentially in a change file. However, in certain conditions (e.g. joins), change files can prove a heavy overhead if they are allowed to grow, since they have to be searched separately on many occasions – see Section 4.3 below.

The tradeoff here is essentially one of frequency of index updating; this could vary from yearly down to daily, and possibly even to more frequently (e.g. triggered by the change file reaching a certain size).

A possible model is $D_{max} = \frac{T_{max}}{vFt}$

where $D_{max}$ = maximum number of days before reindexing
$T_{max}$ = maximum allowable increase in search time due to having to scan change file
$v$ = volatility rate, i.e. proportion of records in file added, modified (affecting indexes) or deleted per day
$F$ = file size
$t$ = CAFS scan time (as in Section 4.1)

Assuming $T_{max} = 1$ s, and a file size itself searchable in 1 s, then $D_{max} = 1/v$, giving 10 days for 10% volatility, 100 days for 1% and so on. If the file is 100 times larger (e.g. 280 Mbytes on FDS 2500), then $D_{max}$ is 100 times smaller – i.e. 1 day for 1% volatility. If the file is used in joins, then only 1/10 or even 1/100 s extra may be acceptable. Compression of change files may help, but the message is that the equations on change file size should always be worked out where volatility is high and deferred index updating is practised.

*Sequence* is an issue in volatile applications because programs inserting new records can quickly find that there is no room in the normal ISAM physical address and are forced into putting the records into overflow. This affects the performance of partial CAFS searches. On the other hand, a lower packing density could slow down the overall rate of searching. As above, the solution has to come through frequent reorganisations, which may be expensive as they are likely to involve rebuilding any associated secondary indexes as well.

*Multiple-valued fields* may be introduced into a file structure used for CAFS searching; this can be done either by denormalising 'child' entities, thus introducing repeating groups, or through coding of text in SIF (see Section 4.5 below). An example would be qualifications and universities attended for persons on a personnel file. A test for 'field-name = value' will result in a hit if any of the occurrences of that field within a record have the correct value. In volatile applications, however, there is a risk that some updates (e.g. new qualifications in the example) may involve increasing the number of occurrences of certain fields, hence lengthening the records. Since records *cannot be fragmented*, this means they have to be regenerated in another place, possibly in overflow or a change file, and wasted space may be left behind. This adds further to the build-up of data needing reorganisation.

Of course there are other problems with multiple-valued fields. A search for QUALIFICATION = BA AND UNIVERSITY = CAMBRIDGE would find anyone who had a BA from *any* university, as long as they had done *something* at CAMBRIDGE. To avoid this, some concept of corresponding values has to be introduced. Another famous condition is QUALIFICATION NOT = COBOL which would retrieve everyone who held a qualification *other* than COBOL, even if they *did* also have COBOL.

*Duplicated data from 'parent' entities* is often recommended for CAFS search optimisation, to save cross-reference look-ups and increase the searching achievable in one CAFS pass. An example would be holding customer details in each order record in a file of sales orders. The implication of this on updating is in terms of what happens when the parent entities themselves get updated. In some cases they are relatively static, but others that are less so could result in the same change having to be repeated in all the records that contain that duplicated data. So a possible message seems to be don't collapse volatile parent entity types!

In connection with this last point, if an associated TP subapplication contains quite complex access paths, it may be too damaging to update performance to force it to work on a CAFS-optimised file structure. If up-to-the-second correctness of query output is not vital, it might be better to decouple the retrieval subsystem and run an extraction process to set up the denormalised CAFS file from the normalised TP database.

### *4.3 Making relational joins between multiple CAFS files*

In some situations it may be impossible to force a denormalised, single-record-type file structure. The designer could be left with several files, each CAFS-searchable, but also needing to be related. This is by no means a rare situation in practice – most commercial systems have several different major entity-types.

The 'pure' CAFS approach would consist of first finding the records qualifying from a search of the first file, and from these ascertaining the

conditions (e.g. matching keys) to be applied to the second file. The second file then has to be searched several times, each for a different set of key matching criteria. With Querymaster this involves a separate scan of the second file for *each* key found in the first; ADMS is rather more efficient, throwing up to 15 keys at a time at the second file with an implied 'OR'.

Practical experience suggests that the volume of 'intermediate results' in relational joins can be very variable indeed, and can certainly not be relied on to be in single or even double figures only. Some further performance assistance is therefore virtually always necessary if the files are much above the 100 kbytes level. Laboratory versions of CAFS included the 'file correlator'[4] for this specific purpose, but this is not yet committed as an enhancement to CAFS-ISP.

At present, then, the designer is thrown back once again onto the use of indexes. To some extent it may be possible to limit this by having the most frequently used match key as the prime key, but this may clash with search partitioning optimisation or updating optimisation. A technique when there are just two match keys is to duplicate the whole file, with one copy with each of the match keys as prime keys, but for too many keys this could become expensive in storage. Secondary keys may already be needed for other search reasons, and match keys may have to be included as additional secondary keys.

In volatile applications there can be further problems when joins are involved, because some records that should be included may not yet be indexed and have to be searched for in change files. If the volume of intermediate results in a join is higher, the penalty of searching the change files a large number of times could be unacceptable. Thus the issue comes back to that of frequency of updating indexes, which was discussed earlier.

### *4.4 Using CAFS searching on IDMS databases*

CAFS searching can be applied to IDMS databases for two main reasons:

- to simplify the physical structure (e.g. by dropping indexes, access path support sets and 'processing convenience' record types) and hence to improve update performance
- to improve retrieval performance on unanticipated ('longstop') query types on specific record types

IDMS has some limitations with respect to CAFS searching: it does not automatically prevent variable-length records from becoming fragmented, so searching where fragments exist may be inaccurate although one is informed; it does not have a mechanism for limiting CAFS searches as in the case of the ISAM primary key, other than by using page ranges. Also, the IDMS common practice of mixing record types on a page in clusters (where member

records are stored 'VIA' a set linking them to the owner records) may be less suitable for CAFS searching than a single-record type AREA or series of AREAs.

However, there is a big advantage in keeping a file in need of CAFS searching within IDMS – namely that Querymaster can use the other sets and indexes for carrying out searches involving several record types, and will use CAFS searching where its rules indicate that this would be advantageous. This avoids many of the problems mentioned in Section 4.3 above.

Having decided to make a record-type searchable by CAFS, most design issues are mandatory[6]. The arguments over denormalisation have already been discussed. Some possible partitioning of the record type into a number of separate areas or page ranges on such bases as live against historical, date of transaction, alphabetic range or geographical grouping could be appropriate. An example would be partitioning insurance policies by period of cover. Dropping of indexes or access path sets will depend on the total data volume and the response requirement.

Although the decision to use CAFS searching on an IDMS database is essentially a physical one and would be presumed to relate to specifications in the Storage Schema rather than the Schema, there are some design issues connected with CAFS usage that do affect Schema design. The chief example is denormalisation. Since the Schema-to-Storage Schema mapping does not allow restructuring, any denormalised structures must be defined in the Schema. Thus there may be differences between the Schema data structure and the 'natural' mapping of the Conceptual Schema.

A final point concerns the use of IDMS in conjunction with CAFS-searchable ISAM files. This may be appropriate if the database contains one very high volume record type on which unpredictable searching is required. If update access paths are fairly simple, then such a record type might better be taken outside the IDMS database. Possible examples could occur where there is a large volume of insurance policies, health-care patients or members of some association, with transaction data for only a small proportion of the population.

### *4.5 Using CAFS for text searching*

Text searching appears to be a fruitful application for CAFS searching for a number of reasons:

- it is a 'natural' storage and retrieval system – arrival of data involves little processing other than making it available for searching
- it is normally single-record-type – i.e. a collection of 'documents' or other bibliographic objects
- the text is relatively easily handled by the CAFS SIF (self-identifying format), together with markers for delimiting the scope of text structure units such as paragraphs, sections etc.

Applying the lessons of Section 4.1 above, some form of secondary indexing will still be required in conjunction with CAFS, both for estimating the number of hits and for giving acceptable response on low hit-rate searches against very large databases. With text the hit rate per byte of file is likely to be lower than for formatted data. For example, a user looking to obtain ten documents from a database of 10 000 abstracts each of an average of 100 words × six characters is creating a hit rate of 1 in 600 000 bytes, which is well below the critical level proposed in Section 4.1.

On the question of updating, one suggested approach is to update the index relatively infrequently and maintain a change file under the guise of an accessions pool. This pool would be nonindexed and would be CAFS searched *before* the main file, since more recent documents might well be of more interest. The user could be given hits for the pool and for the main file (via the indexes) separately. The tradeoff here is how large to keep the pool and how to 'housekeep' older accessions into the indexed main file.

Even with full indexing on text words, there will still be some queries which fall into the longstop category. An example is a fuzzy match when the leading characters are unknown. In this case the indexes will not be of any help, and full CAFS scan will be necessary. In the cases of quorum and weighted searches the indexes may help provide an initial screen of documents having none of the required words, but if the remainder is large it might be better to scan directly – response required would probably not be critical.

The designer may also have to make a choice about the detail held in any indexes, unless their format is predefined by the software. If proximity searching is required, i.e. how many words apart two chosen words appear in a document, then full details of word occurrence position would need to be held. Otherwise less detail could be used – at the extreme only a bit map of qualifying search cells or documents need to be held.

Another choice concerning the index is whether multiple concordances are worthwhile. Some user queries might stipulate that the required words must appear in the title, the abstract, or the full text, whereas others might want documents containing the same words in any position. It has been proposed[7] that separate concordances (i.e. separate indexes for each searchable area of documents) could be used. Certainly these would be efficient if bit maps were used to hold the indexes and the search cells were large enough to keep the bit maps fairly short. The main alternative is to choose different SIF fieldnames for the words according to where they appear, which would enable the criterion of where the word occurs to be evaluated when the search cells pointed to by the main index are scanned.

### *4.6 Avoiding degradation of concurrent TP*

A further design issue which has already received attention is that of how to stop long CAFS scans from causing an unacceptable degradation on TP

response. The main way of solving this problem is by setting a CAFS fragment size to limit how much of the files can be searched before TP operations are given a chance to use the disc channel. Fragments are not the same as search cells, but they may be of similar size (e.g. between 1 track and 1 cylinder). No models or rules of thumb are suggested here – the topic is addressed in Reference 6.

Avoiding TP degradation is also a possible reason for separating the TP subapplication from the retrieval subapplication, if it is acceptable to run retrieval from periodically copied or updated files.

## 5 Summary of design parameters

The lessons of Sections 4.1 to 4.5 are summarised in Table 2.

**Table 2 Design parameters**

| Type of database<br>Parameter | Considerations |
|---|---|
| All database types | |
| Complementary use of secondary indexes | – response times required for anticipated queries and updating<br>– size of file<br>– need for count of hits with successive refinement<br>– relational joining: match keys |
| Size of search cells | – appears to be no 'in-between'<br>– for high hit rates, full search<br>– for low hit rates, indexing to block |
| Frequency of index updating and change file housekeeping | – volatility of file<br>– need for up-to-second correct answers<br>– level of joined queries using indexes<br>– cost of indexing runs |
| Use of multiple-valued fields | – Risk of extension of record length, hence fragmentation or overflow<br>– frequency of queries involving both these fields and single-valued fields |
| Holding duplicated 'parent' data | – volatility of parent entity types<br>– frequency of queries involving both these fields and 'home-entity' fields |
| ISAM databases | |
| Choice of primary key | – useful partitioning of searching i.e. starting/finishing at given key values<br>– main key used in TP updating<br>– match key used in relational joins |
| Duplicating files involved in several joins, using different primary keys | – cost of storage and keeping in step<br>– cost of secondary indexing<br>– response requirements on joined queries |
| Frequency of reorganisation | – volatility of file<br>– frequency of queries partitioned by primary key values<br>– cost of reorganisation runs |

| | |
|---|---|
| ISAM against IDMS | – variable length, volatile data<br>– likelihood of partitioned searches<br>– cross-referencing to different record types |
| IDMSX databases | |
| Dropping of IDMS access-path sets, entry point records, IDMSX indexes<br>Use of multiple IDMSX AREAs for record type | – volume of records<br>– response time requirements<br>– likelihood of partitioned searches<br>– progression of data from live to historic |
| Text databases | |
| Text 'accessions pool' | – weighting of interest towards most recent accessions<br>– (see indexing/change file updating) |
| Detail of indexing on text words<br>Use of separate concordances | – need for proximity searching<br>– frequency of searches on words in specific sections of document<br>– size of index entries<br>– hit rate |
| Databases with TP input mode | |
| Size of SEARCH-FRAGMENT | – response requirement of concurrent TP on same file |
| Decoupling CAFS search file from TP database | – volatility of file<br>– complexity of TP processes<br>– cost of storage and copying<br>– need for up-to-second correct answers |

## 6 General conclusions

It should be clear from the length of the list in Section 5 above that, to get the best use of CAFS, a designer may have to take a number of parameters into account. Although some physical structures may be rendered unnecessary in certain cases, it seems likely that most existing aids to performance will continue to be used in conjunction with CAFS.

The opportunity to drop all access path sets and secondary indexes is only really an option up to file sizes of 10 Mbytes (or 28 Mbytes on the new FDS 2500 discs) unless response times on single file searches of over 10 s are permitted. Where joins are involved in answering queries, even files down to 100 kbytes may still need to be indexed to avoid response being slowed by high volumes of intermediate hits. Indexes are also likely to be valuable for providing counts of expected hits to the user.

Equally, even with a fairly high level of indexing, there is still much to be gained by the use of CAFS, particularly where the search is fuzzy or of a type where the indexes do not provide much help, or where queries on a particular field are so rare as not to justify the cost of storing and maintaining the extra indexing. It is perhaps this order of magnitude improvement in 'longstop' cases where CAFS will show the greatest gain for large files.

**Acknowledgment**

This paper arose from work done in a consultancy capacity when evaluating a CAFS-based proposal for a client. The assistance of client personnel and the ICL marketing staff involved is acknowledged. Acknowledgments are also due to Hamish Carmichael and Guy Haworth for discussions, comments and the provision of useful documents and again to Guy Haworth for assistance on the timing formulae and the figures for the graphs.

**References**

1 CARMICHAEL, J.W.S.: 'History of the ICL content-addressable file store (CAFS)', *ICL Tech. J.*, 1985, **4** (4), 352–357.

2 HAWORTH, G.McC.: 'The CAFS system today and tomorrow', *ICL Tech. J.*, 1985, **4** (4), 365–392.

3 CROCKFORD, L.E.: 'Associative data management system', *ICL Tech. J.*, 1982, **3** (1), 82–96.

4 MALLER, V.A.J.: 'The content addressable file store – CAFS', *ICL Tech. J.*, 1979, **1** (3), 265–279.

5 WILES, P.R.: 'Using secondary indexes for large CAFS databases', *ICL Tech. J.*, 1985, **4** (4), 419–440.

6 ICL Computer Users Association (UK), CAFS User Group: 'Exploiting CAFS-ISP', Working Party Report, July 1984 (2nd Amended Reprint, July 1985), ICLCUA (UK), PO Box 42, Bracknell, Berks. RG12 2LQ, UK.

7 CARMICHAEL, J.W.S.: 'Application of ICL's content addressable filestore to text storage and retrieval', in *'Protext I: Proceedings of the first international conference on text processing systems'*, MILLER, J.J.H. (Ed.), Boole Press, Dublin, 1984.

# Using secondary indexes for large CAFS databases

**P.R. Wiles**
ICL Inland Revenue Project, Telford, Shropshire

**Abstract**

The Inland Revenue have implemented a pilot CAFS-based system for tracing taxpayers from their names and addresses. This system is in live use on a taxpayer database of around 4 million records. It has now been decided to extend this system to cover the whole country, which will mean an increase of the database size to around 48 million records, and a major increase in the number of searches which must be supported per second. This workload is outside the realms of possibility with the existing CAFS hardware and software, and the paper discusses the use of secondary indexing techniques to reduce the number of blocks which must be searched by CAFS per enquiry.

## 1 Introduction

The Inland Revenue receives a large amount of correspondence from taxpayers and outside bodies. A lot of this is 'unreferenced', i.e. it does not contain the reference number of the taxpayer to whom it refers. On the previous manual system this reference number for PAYE taxpayers was the 'Employer Reference'. In the new computerised PAYE system it is the National Insurance Number (NINO).

Under both systems it is very difficult to associate this correspondence with the relevant taxpayer, and time-consuming manual methods are currently employed to trace the associated reference number. Thus the Inland Revenue has investigated the use of computerised techniques to trace taxpayers from the information which is normally available in this correspondence – principally the taxpayer's name and address.

A CAFS-based system has been implemented to provide this function. It enables a tax officer to identify the NINO, and the responsible tax district, of any taxpayer whose name and address contain a given set of words. The system is currently in use at the Inland Revenue's Centre 1 office in Scotland. This computerised centre has been in use for some years, and thus it is a suitable site for a pilot tracing system.

The Centre 1 tracing system contains details of around 4 million taxpayers, and handles a peak workload of around one trace every two seconds. It runs on an ICL 2966 with a single CAFS unit.

An online transaction within a TPMS (transaction processing) service provides the tracing service. It displays a screen format onto which the tax officer enters those details of the name and address that he or she has and judges will be most effective in the search. These details are used by the tracing application code to initiate a CAFS search on the tracing database. If any taxpayers are found by this search, their details are displayed on the same screen format. If there are too many to fit onto one screen a 'paging' mechanism is used to display later ones once the tax officer has seen the earlier entries.

The data required in the tracing database is held in a large Indexed Sequential (ISAM) file. The key of each record consists of the first five characters of the taxpayer's surname, followed by his or her NINO, followed by other characters to ensure uniqueness. The average length of a record is 150 characters.

The application code invokes the ISAM record access facilities, specifying a search program which would normally include an indication of part or all of the prime key of the file (specifically the taxpayer's surname). This key is used to determine the range of blocks in the file which contain all the entries for the given surname or surname stem. These blocks are then searched by CAFS and any records which contain the specified surname and address words are returned to the application.

The tax officer may specify 'fuzzy' words (e.g. 'B?AT', which may be satisfied by 'BOAT', 'BEAT', 'BRAT' and so on). In addition, if a search fails to find a single address, he may specify a 'quorum', saying (in effect) 'well, can you find any address which satisfies three out of the four words I specified?' Both of these facilities are mapped directly on to CAFS facilities by the tracing application software.

It is rarely necessary to enter complete words. The first few characters of the more significant words in the address are usually more effective in providing an accurate match than a complete surname and a complete address word. This saves keying time for the user, as well as minimising the effect of inconsistency of spelling between the stored records and the tracing information. Experience with the pilot system shows that, after a relatively short experience, users become adept at deciding how much or how little to enter, and at picking out which terms within a name and address will provide the best discrimination.

## 2 National tracing system

Replication of the COP system is now under way across all tax districts, and the Inland Revenue wish to extend this tracing facility to all the tax officers who have access to the computerised systems. This will involve an order-of-magnitude increase in both the volume of data held and the number of messages which must be handled by the tracing system, compared with the

pilot system. The Inland Revenue expects that, by the time the complete PAYE system is in place, the national tracing system will be expected to handle 15 name-and-address tracing messages a second. This includes the messages arising from the recent decision to add Schedule D (self-employed) taxpayers to the COP system.

The tracing database in a national tracing system is expected to contain 48 million records, with an average record size of 150 bytes. With a packing density of 85%, this amounts to just under 8500 Mbytes of data.

Normally, large files such as these would be placed on FDS 2500 discs. However, these files are to be accessed via CAFS, and sizing studies show that the target message rate will give rise to very high disc utilisations which can be alleviated by using FDS 300 discs instead. It is expected that the tracing database will be duplexed, and if so it will occupy 64 FDS 300 discs in all.

A block size which allows six blocks to a track is used. This is a compromise between the small block sizes required for random accesses to the file and the larger block sizes which optimise serial processing, for example in a batch updating job.

The principal constraints on the message throughput rate will be the disc utilisation and controller utilisation. Controller utilisation is not normally an important factor but when the controller includes a CAFS unit it may become so. It is important to use as many CAFS units as possible; i.e. one for every two discs.

With the configuration derived from these parameters, it is clear that it is not possible to allow each tracing enquiry to search, on average, more than 72 contiguous tracks on an FDS 300 disc. Yet sizing studies show that, using a four-character surname stem, the ISAM RAM will on average search 1238 tracks per enquiry. Thus to meet the required message throughput rate it is necessary to significantly reduce the number of tracks searched by CAFS per enquiry. This is to be achieved by a new software development, currently called the Advanced CAFS Option (or ACO), which offers a secondary indexing capability for CAFS databases.

## 3 Using secondary indexes

### *3.1 Indexing large data files*

A CAFS data file contains a number of records, each of which holds a number of fields. These fields may be identified either by their position within the record, or by a 'self-identifying format' (SIF) field number, which can be recognised by the CAFS unit. Fixed fields occur once and only once in each record. SIF fields may occur none, once or more than once in a record.

Any field in a record may be used to identify one or more records within the file which hold a specific value in that field. One field in particular can be

chosen as the primary key, and the file as a whole may be sorted on the value held in that field. It is sensible if this is a fixed field, which must occur once and only once in each record.

Any field within the record may be used to construct an index to the file. An index lists all the values which occur in that field (the keywords), and identifies those records which contain each specific keyword. If the file is ordered by that field, then this is the primary key to the file. The standard indexed sequential key is an example of a primary key. The primary key is usually selected such that each value of the key identifies a single record within the file.

A secondary index is an index based upon some field other than the primary key. It may identify all the records which hold specific values in a specific fixed field – the Alternate Key index to an Indexed Sequential file is an example of this – or it may be less precise, indicating merely the portions of the file in which the records with particular values lie.

It is relatively easy to ensure that each value of a primary key identifies a unique record. It is much more difficult to ensure the same about a secondary key, and in general is not worth the trouble.

A secondary index which indicates the range of records within the file holding instances of particular field values is particularly useful in conjunction with CAFS, as it may be used to restrict the portion of the file over which the power of the CAFS search facility may be applied. For example, it is possible to consider the file to be made up of a number of equal-sized search units (blocks or multiples of blocks) and construct a secondary index which identifies the set of search units holding any particular value in a particular field. The effectiveness of such an index may be varied by changing the size of each search unit, or the portion of the field value that is contained in the index (the stem size), or even the block size or packing density of the file. A larger stem size, or a smaller search unit size, will lead to a more effective (but bigger) index; a smaller stem or a larger search unit size leads to a coarser (i.e. less effective) index but a smaller and hence more manageable one.

A conventional secondary index must identify specific records, and thus tends to be very large but very effective. A secondary index as is proposed to be used here relies upon the power of CAFS to identify the required records from a relatively small search area. The index itself can thus be less effective.

A field which is used in a secondary index should be one which offers a reasonable number of distinct values, especially if a search unit is large enough to contain many records. Thus it is possible to construct secondary indexes for the Inland Revenue tracing database based upon, for example, surnames, words which occur in addresses, postcodes or parts of postcodes, and so on. A single secondary index may contain values from a number of different fields; this is especially useful given that an address as held in the

tracing database consists of four different SIF field values corresponding to the four lines available for the address in Inland Revenue correspondence. Thus by using a secondary index based upon the taxpayer's address, it is possible during a tracing enquiry to read the secondary index entry for each significant word or word stem specified by the tax officer, and from this obtain for each word the set of search units which contain that word. The CAFS search can then be confined to those search units which are obtained by taking the intersection of all these individual sets. Suppose, for example, that a tax officer instigates a search over a rather small example database for:

Surname: WILES Address: LAWN CENTRAL TELFORD

The Indexed Sequential file index may indicate that surnames starting with WILES occupy search units 190–205 in this file. A secondary index may indicate that, taking four-character stems, the three address words occur in (among others) the following search units.

| | |
|---|---|
| LAWN | 190, 194, 195, 200, 201, 202, 204, 205 |
| CENT | 190, 191, 194, 196, 197, 200, 201, 203, 205 |
| TELF | 190, 191, 194, 195, 200, 204 |

Thus a CAFS search which is confined to the intersection of these three sets will only scan search units 190, 194 and 200. This amounts to three search units, whereas a search determined solely by the surname would scan 16 search units.

Fig. 1 illustrates the way in which a secondary index is used with an Indexed Sequential file.

The tax officer may also specify a quorum, used in the normal CAFS sense, and the computation of the set of units to search is then more complex than merely taking the intersection of each keyword's search unit set.

For example, if A, B and C are the sets of search units corresponding to three keywords, the simple computation if all three terms must appear in the chosen search units is:

A and B and C

whereas if it is sufficient to find units with any two out of the three, then the computation is:

(A and B) or (B and C) or (C and A)

Extending the example used above, a search for WILES and any two out of LAWN, CENTRAL, TELFORD would scan search units:

190, 191, 194, 195, 200, 201, 204, 205

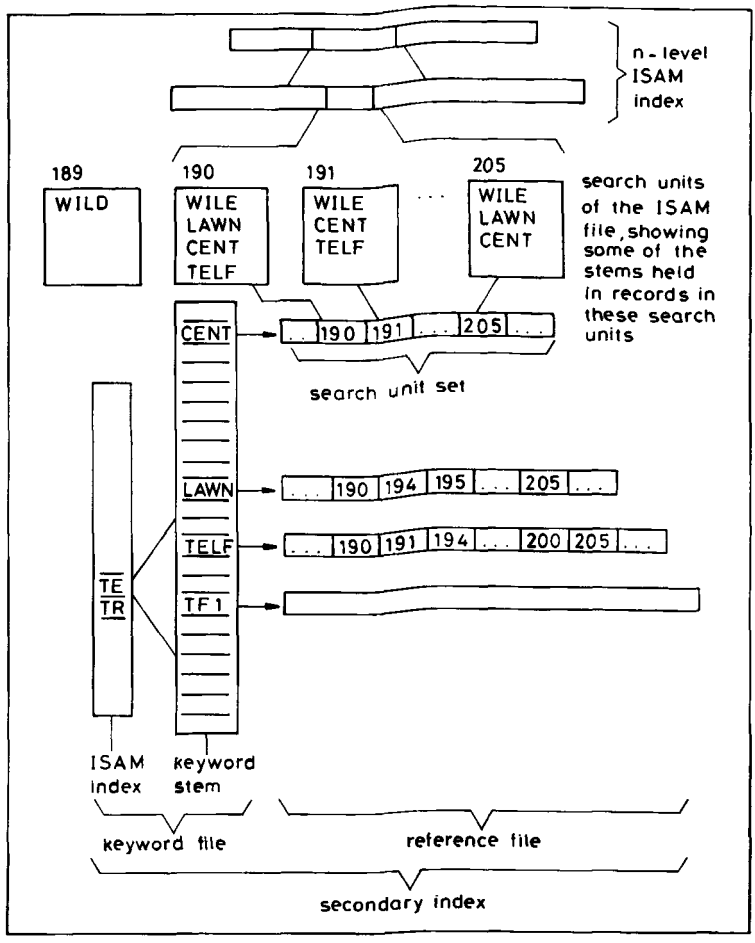


Fig. 1 Layout of a file with a secondary index

or eight units, rather than the 16 specified by the ISAM index alone. Thus using a quorum will give a shorter search than if no address keywords were used at all, but it is considerably worse than if a simple and condition is used, and will take a great deal more computation to determine the search unit set.

Names of types of road (CLOSE, AVENUE, STREET, ROAD, LANE etc) would not normally be required as part of an address-based secondary index, although some (CLOSE and LANE for example) could be significant in a surname-based index. For simplicity of creating the secondary indexes, it is best to handle all words in the address as significant, and set them up as keys to the secondary index. The index entries for these words would identify a great many (possibly all) search units, which would make them useless in

restricting the size of a search. However, a skilled tax officer would not use such common words in a tracing enquiry, and thus their presence in the secondary index would have no significance other than the space that they occupy (which would be negligible).

### *3.2 Characteristics of an address-based secondary index*

An important characteristic of any secondary index is the size of the stem used to access the index. This is a function of the number of distinct values of the field upon which the index is based – in this case on the total number of different words occurring in taxpayer's addresses. It is also a function of the ways in which these words are distributed across the population. It is obvious that some words occur in many more addresses than others.

Information which illustrates this distribution is derived from a file containing a list of all the taxpayers handled by Centre 1. This file (the Centre 1 Index) contains about 2·16 million addresses. Each address contains on average 5·85 words. There are 197 523 distinct words in these addresses. About 40% of the words contain numeric characters, and are thus house or flat numbers or parts of postcodes.

**Table 1 Cumulative-frequency table for Centre 1 Index**

| Percentage of stems | Maximum addresses holding any given stem | | | | |
|---|---|---|---|---|---|
| | stem = 20 | stem = 8 | stem = 5 | stem = 4 | stem = 3 |
| 100 | 398 698 | 398 698 | 399 056 | 401 186 | 412 844 |
| 99 | 609 | 688 | 1644 | 3511 | 10 418 |
| 98 | 286 | 303 | 605 | 1394 | 4407 |
| 97 | 214 | 228 | 345 | 695 | 2250 |
| 96 | 149 | 167 | 274 | 411 | 1232 |
| 95 | 102 | 117 | 234 | 317 | 737 |
| 90 | 26 | 30 | 76 | 180 | 277 |
| 85 | 11 | 12 | 29 | 72 | 219 |
| 80 | 6 | 6 | 14 | 33 | 142 |
| 75 | 4 | 4 | 8 | 18 | 70 |
| 70 | 3 | 3 | 5 | 11 | 36 |
| 65 | 2 | 2 | 4 | 7 | 19 |
| 60 | 2 | 2 | 3 | 5 | 12 |
| 55 | 1 | 1 | 2 | 4 | 8 |
| 50 | | | 2 | 3 | 5 |
| 45 | | | 2 | 2 | 4 |
| 40 | | | | 2 | 3 |
| 35 | | | | 1 | 2 |
| 30 | | | | | 2 |
| 25 | | | | | 2 |
| 20 | | | | | 1 |

The number of occurrences of each distinct stem in this file, for various stem sizes, is shown in Table 1 in the form of a cumulative-frequency table. This shows first of all the number of addresses which hold the most common stem.

100% of all stems occur in this number of addresses, or less. It then shows that 99% of all stems occur in many fewer addresses each, and so on. Thus, for example, for a four-character stem:

| | |
|---|---|
| 50% of all stems occur in | three addresses or fewer each |
| 95% of all stems occur in | 102 addresses or fewer each |
| 99% of all stems occur in | 3511 addresses or fewer each |
| 100% of all stems occur in | 401 186 addresses or fewer each. |

Most interestingly, only 1% of all distinct four-character stems occur in more than 3511 addresses each, and no stem occurs in more than 401 186 addresses. The stem which occurs in the most addresses is GLAS, which is in 18% of all addresses in the Centre 1 Index.

Of the various stem sizes considered, a stem of four characters appears to be of the most use. It is the largest stem which it is practical to require a tax officer to type, and yet it gives a significant discriminatory power – over twice that of a three-character stem, for instance.

The total number of distinct stems, for different stem sizes, is of great importance to the eventual size of a secondary index. This information is not available directly. Fig. 2 shows how the numbers increase with the population size, and is based upon data extracted at various points while analysing the Centre 1 Index. However, these graphs do not allow extrapolation to the numbers which will be found across the entire country. The absolute maximum number of four-character stems is:

$$36^4 + 36^3 + 36^2 + 36$$

or 1 727 604. This includes all combinations of up to four letters and digits. Most of these combinations will not occur in practice, even in postcodes. If it is assumed that an address word stem contains at least one vowel, and that a postcode component contains one or two letters and one or two digits, then the useful maximum limit for four-character stems is around 200 000. This figure will be used in sizing the Inland Revenue tracing system.

Examination of the numbers of occurrences of selected words in the Centre 1 Index, in different sized samples of addresses, shows that the ratio

$$\frac{\text{number of occurrences of word in sample}}{\text{number of addresses in sample}}$$

for any given word is reasonably constant whatever the sample size. It is thus reasonable to extrapolate from the number of occurrences of a particular word in the sample represented by the Centre 1 Index (around 2·16 million addresses) to the number of occurrences of that word to be expected in the full National Index of 48 million addresses (although there will be exceptions to this, such as GLASGOW or words beginning with MAC which can be expected to be particularly prevalent in a Scottish-based index).

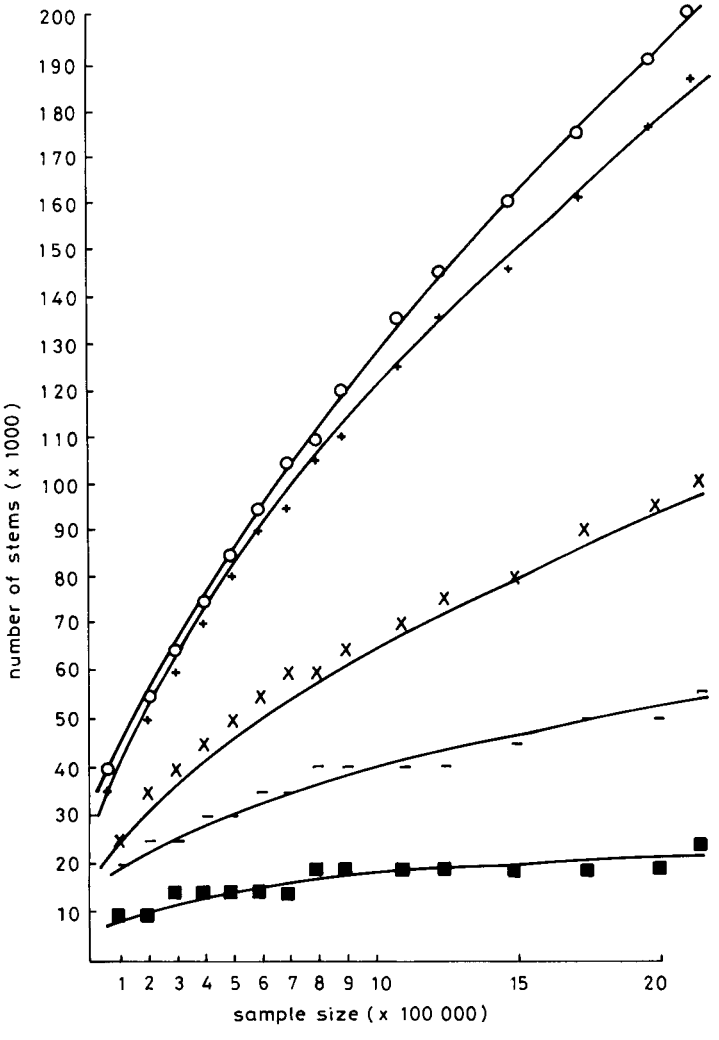


Fig. 2 Tendency of numbers of distinct stems with sample size
Key
0: 20-character stems
+: eight-character stems
×: five-character stems
−: four-character stems
■ three-character stems

Table 1 shows the way in which different words occur in different numbers of addresses. It is necessary to determine the corresponding spread of words across search units, as this will determine the effectiveness of a secondary index based upon address words.

Let:

$TK$ = number of distinct stems occurring in one or more records

$t$ = number of records containing the most common stem

$n(i)$ = number of stems which occur in just $i$ records (for $i = 1, 2, 3, \ldots, t$)

$p(i) = \dfrac{n(i)}{TK}$ for $i = 1, 2, 3, \ldots, t$

$TU$ = total number of search units in data file.

It is required to determine the number of search units which contain a keyword that occurs in $i$ records. If all the units contain the same number of records and the records are distributed randomly over the units, the probability that a particular unit will contain a particular record is $1/TU$; and therefore the probability that it does *not* contain that record is $1 - 1/TU$. Therefore the probability that the unit contains none of the $i$ records in which the keyword occurs is

$$(1 - 1/TU)^i$$

since they are distributed independently; and the probability that it contains at least one of these records is

$$1 - (1 - 1/TU)^i \tag{1}$$

Thus the number of search units containing a keyword which occurs in $i$ records, $g(i)$, is:

$$g(i) = TU\left\{1 - \left(1 - \frac{1}{TU}\right)^i\right\} \tag{2}$$

Eqn. 2 can be used to estimate the number of these search units holding occurrences of a given keyword as the number of records containing that keyword varies.

It is estimated that the national tracing database will occupy 1 411 765 blocks of 6032 bytes on FDS 300 discs. Table 2 shows how varying numbers of records map onto numbers of search units with this database size, for search unit sizes of 6, 3, 2 and 1 blocks. It also shows each number as a percentage of the total number of search units for that search unit size.

**Table 2 Proportions of search units corresponding to proportions of records**

| Number of records in which stem occurs | Percentage of total records | Search unit size, blocks 6 | | 3 | | 2 | | 1 | |
|---|---|---|---|---|---|---|---|---|---|
| | | search units | percentage of 235 294 | search units | percentage of 470 588 | search units | percentage of 705 883 | search units | percentage of 1 411 765 |
| 1000 | 0·002 | 998 | 0·42 | 999 | 0·21 | 999 | 0·14 | 1000 | 0·07 |
| 10 000 | 0·021 | 9790 | 4·16 | 9895 | 2·10 | 9930 | 1·41 | 9965 | 0·71 |
| 100 000 | 0·208 | 81 466 | 34·62 | 90 089 | 19·14 | 93 240 | 13·21 | 96 541 | 6·84 |
| 200 000 | 0·417 | 134 726 | 57·26 | 162 932 | 34·62 | 174 163 | 24·67 | 186 479 | 13·21 |
| 300 000 | 0·625 | 169 546 | 72·06 | 221 830 | 47·14 | 244 398 | 34·62 | 270 268 | 19·14 |
| 400 000 | 0·833 | 182 310 | 81·73 | 269 452 | 57·26 | 305 355 | 43·26 | 348 327 | 24·67 |
| 500 000 | 1·042 | 207 192 | 88·06 | 307 957 | 65·44 | 358 261 | 50·75 | 421 048 | 29·82 |
| 600 000 | 1·250 | 216 922 | 92·19 | 339 091 | 72·06 | 404 178 | 57·26 | 488 796 | 34·62 |
| 700 000 | 1·458 | 223 283 | 94·90 | 364 265 | 77·41 | 444 030 | 62·90 | 551 911 | 39·09 |
| 800 000 | 1·667 | 227 442 | 96·66 | 384 620 | 81·73 | 478 618 | 67·80 | 610 710 | 43·26 |
| 900 000 | 1·875 | 230 160 | 97·82 | 401 077 | 85·23 | 508 637 | 72·06 | 665 489 | 47·14 |
| 1 000 000 | 2·083 | 231 938 | 98·57 | 414 384 | 88·06 | 534 691 | 75·75 | 716 521 | 50·75 |
| 2 000 000 | 4·167 | 235 246 | 99·98 | 463 875 | 98·57 | 664 365 | 94·12 | 1 069 382 | 75·75 |
| 3 000 000 | 6·250 | 235 293 | 100·00 | 469 786 | 99·83 | 695 814 | 98·57 | 1 243 154 | 88·06 |
| 4 000 000 | 8·333 | 235 294 | 100·00 | 470 492 | 99·98 | 703 441 | 99·65 | 1 328 730 | 94·12 |
| 5 000 000 | 10·417 | 235 294 | 100·00 | 470 577 | 100·00 | 705 291 | 99·92 | 1 370 873 | 97·10 |
| 10 000 000 | 20·833 | 235 294 | 100·00 | 470 588 | 100·00 | 705 883 | 100·00 | 1 410 581 | 99·92 |

As is to be expected (and is shown elsewhere in this issue of the *ICL Technical Journal*), Table 2 shows the greater discriminatory power of a smaller search unit size. With a size of one track, a keyword need only occur in 2% of all records before it becomes relatively unusable for refining a search. If the search unit size is one third of a track, it may occur in 6% of all records before it becomes relatively useless.

Note that Table 1 shows that the most common word in the Centre 1 Index (GLASGOW) occurs in 18% of all addresses in that (admittedly atypical) file, and hence would be expected to occur in all search units whichever of the search unit sizes was used.

Eqn. 2 assumes that the distribution of records across search units is completely random. In practice, of course, there is only room for a limited number of records in each search unit. Thus the actual number of search units containing a particular keyword will be slightly higher than the value given by the formula, but the difference is not significant.

Table 1 shows that, for the Centre 1 Index and using a four-character stem, 50% of all distinct keywords occur in three or less records. However, because of their rarity, these keywords will not be given in as many tracing requests as the more common words, which occur in more records. It is much more useful to determine the average number of search units containing those words which actually occur in tracing enquiries.

In other words, given a random keyword in a record chosen at random from the tracing database, what is the average number of records that contain that keyword?

The total number of keyword stem instances ($TS$) is

$$TS = 1.n(1) + 2.n(2) + 3.n(3) + \ldots + t.n(t)$$

$$= \sum_{i=1}^{t} i.n(i) \qquad (3)$$

This figure can also be deduced by multiplying the total number of records (48 million in the case of the national tracing database) by the average number of keywords per record (5·85). It is expected to be about 281 million for the tracing database.

Call a keyword a class $i$ keyword if it occurs in exactly $i$ records. Then, given a random keyword instance,

$$\text{Prob (keyword is class } i) = \frac{\text{number of class } i \text{ instances}}{\text{total number of instances}}$$

$$= \frac{i.n(i)}{TS}$$

Consequently, the mean number of records containing a random keyword instance is

$$= \sum_{i=1}^{t} i.\text{Prob (keyword is class } i\text{)}$$

$$= \frac{1}{TS} \sum_{i=1}^{t} i^2.n(i) \qquad (4)$$

The values of $n(i)$ approximate to an inverse exponential distribution. However, the tail is too long to be able to use the characteristics of such a distribution. The tail in fact consists of isolated values of 1 (corresponding to the numbers of occurrences of the few most common words) interspersed with many zeroes. The values for these common words (large values of $i$) have a much more significant effect on the mean value than the terms for the less common words. These common words, and their distributions in the Centre 1 Index, are known. Assuming that the distribution will be the same in the national tracing database, then summing these last terms of eqn. 4 in reverse with the values of $i$ extrapolated to the number of occurrences expected in the whole country indicates that the mean across the whole country should be less than 1 626 000.

Put another way, this implies that if a random word is taken from a random tracing request, on average fewer than 1 626 000 other addresses in the tracing database may be expected to hold that word.

*3.2.1 Effect of surnames:* Whereas Table 1 shows that the average four-character address stem occurs in three addresses or fewer, surnames have a smaller distribution and the average four-character surname stem occurs in 72 records, out of a sample size of 2·16 million records. However, as with address stems, the likelihood that a surname will be given in a trace is proportional to the number of instances of that surname, and hence the average number of surnames identical to a surname given in an average tracing request is higher than this figure.

Summing eqn. 4 in reverse for the most common surname words shows that the mean number of occurrences of a surname chosen from a random record is expected to be less than 252 000 across the whole country. As the tracing database is ordered on surname, these records will occupy a range of contiguous blocks, and with the block size, record size and packing density used this will occupy 1238 FDS 300 tracks, taking around 20 s to scan.

This is thus the average number of tracks which would have to be searched if the first four characters of the surname are used to delimit the search area across a national tracing database ordered on surname, using the same tracing techniques as in the Centre 1 system. The use of a secondary index based on address words will enable this figure to be reduced to a more manageable number of tracks.

*3.2.2 Structure of a secondary index:* The sizing calculations described in this paper formed the basis for the justification to implement a software product called, tentatively, the Advanced CAFS Option, or ACO. This provides a secondary indexing capability along the lines described above, and will be used by the Inland Revenue tracing application in place of the standard Indexed Sequential record access facilities.

As well as providing these facilities, ACO is designed to be used with very large databases, and it allows these to be split into a number of content files, ideally with each file held on a separate disc. The content files must have contiguous but nonoverlapping key sequences. This means that for tracing purposes the database can be considered as a single monolithic file, but for housekeeping operations it can be treated as a number of distinct files. For the Inland Revenue tracing database, a figure of one content file for every 600 Mbytes or so of data gives a total of 16 files, which allows a reasonable cyclic pattern of housekeeping operations.

The combination of all the content files, any secondary indexes which are used with it and the files which define the relationship between the content files and the structure of the secondary indexes, is called a file complex.

A secondary index as implemented by ACO comprises two parts:

- a keyword file, which contains fixed-length records, each indicating that a particular stem occurs in the database
- a reference file, indicating those search units in the database which hold records containing a particular keyword stem.

*Keyword file:* The key to the keyword file is the keyword stem. Thus the existence of a keyword record shows that there is at least one instance of the stem within the database.

The keyword record indicates the number of search units which contain the keyword, which allows processing of a search program to concentrate on the most effective keywords first. In some cases it can also indicate that these search units are all held in a single content file, and if the set of search units is contiguous within that file it can indicate the precise range. This is most useful if the secondary index mirrors the Indexed Sequential file primary key.

For the Inland Revenue tracing system, the address index keyword file contains around 200 000 records, each of 12 bytes, giving a total size of 2·4 Mbytes.

*Reference file:* Other search unit sets are represented by records in the reference file. There is one such record for each keyword stem for each content file in which the stem occurs. The records are keyed by a combination of the stem and the content file number. The remainder of the record indicates the set of search units within the content file which hold one or more records

containing instances of the keyword stem. There are a number of ways in which this set may be represented, of which two are most relevant.

*Format 1:* In this format each reference file record contains a list of search units, each indicated by its position relative to the start of the content file. Three bytes are used to identify each search unit, and there is an overhead of 9 bytes per record. Thus if a search unit set identifies $u$ search units, and these are spread over $f$ content files, then this format requires:

$$9 \times f + 3 \times u$$

bytes per keyword.

*Format 2:* In this format the search unit set is indicated by a bitmap with one bit for each search unit in the content file. If a search unit contains an instance of the keyword stem, then the corresponding bit is set to 1. As well as the bitmap, there are 15 additional bytes per record. If the entire database is split into $TU$ search units, again spread over $f$ content files, then the total size of the data representing one keyword in this format is approximately:

$$15 \times f + TU/8 \text{ bytes}$$

Obviously Format 1 is most useful where the keyword occurs in only a few search units, and Format 2 where the keyword is more common. The two formats occupy the same space when

$$9 \times f + 3 \times u = 15 \times f + TU/8$$

i.e. when

$$u = TU/24 + 2 \times f$$

or approximately 5% of $TU$.

However it is more efficient to merge search unit sets in Format 2 than in Format 1, and thus ACO tends to use Format 2 for smaller search unit sets than is required solely on space grounds. The default is to use Format 2 where a keyword occurs in more than 1% of search units.

### *3.3 Size of the address reference file*

The keyword file for the address secondary index occupies around 2·4 Mbytes. Compared with a database size of 8·5 Gbytes this is insignificant. However, the reference file will be much larger, and if its size is not considered carefully it may cause significant problems when the database is updated.

The mean number of records holding occurrences of any given keyword, $m_r$, can be calculated as:

$$m_r = \frac{TS}{TK} = \frac{1}{TK} \sum_{i=1}^{t} i.n(i) \tag{5}$$

The corresponding formula for the mean number of search units per keyword, $m_u$, is

$$m_u = \frac{1}{TK} \sum_{i=1}^{t} g(i).n(i) \tag{6}$$

where $g(i)$ is given by eqn. 2.

$m_u$ is difficult to calculate precisely using eqn. 6, but will always be less than $m_r$ (because $g(i)$ is never larger than $i$).

The shape of the graph of $g(i)$ is shown in Fig. 3.

A reasonable approximation to $g(i)$ for sizing purposes is

$g(i) =$ **if** $i > TU$ **then** $TU$ **else** $i$ **fi**

and this is shown in Fig. 4. With this approximation,

$$m_u = \frac{1}{TK} \sum_{i=1}^{TU} i.n(i) + \frac{1}{TK} \sum_{i=TU+1}^{t} TU.n(i) \tag{7}$$

If the address reference file is represented entirely by Format 1, then its size will be

$$TK \times (9 \times f + 3 \times m_u) \text{ bytes}$$

although there will be a wide variation in record sizes, and a low packing density (say around 60%) will be required.

If Format 2 is used throughout, then the occupancy is

$$TK \times \left(15 \times f + \frac{TU}{8}\right) \text{ bytes}$$

A high packing density could be used with the file organised in this way, as the Format 2 records have a fixed length.

The optimum file size occurs if a combination of Formats 1 and 2 is used. It is proposed that any stem occurring in more than 1% of all search units is to be represented by Format 2, and the remaining keywords by Format 1. Again, this organisation would lead to a wide variation in record sizes, and a packing density of 60% has been assumed.

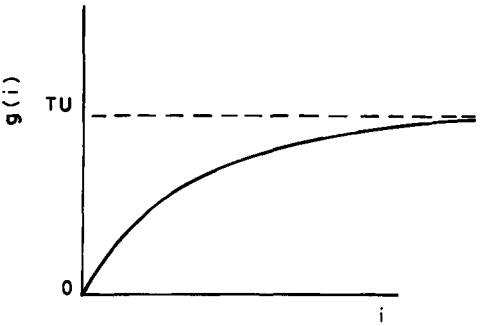


Fig. 3 Graph of $g(i) = TU\left[1 - \left(1 - \frac{1}{TU}\right)^{i}\right]$

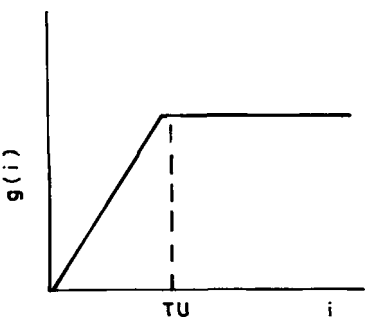


Fig. 4 Graph of **if** $i > TU$ **then** $TU$ **else** $i$ **fi**

Table 3 shows the estimated size of the reference file with these three organisations. It shows the effect of using the optimised approach, but even with this the reference file will occupy a substantial amount of disc space, and furthermore sizing studies show that the number of accesses to it per second require it to be spread over a number of discs.

The complete search unit set for a keyword can occupy (in the case of the Inland Revenue tracing database) 16 records, and up to 88 kbytes of disc space. If all of this data must be accessed by the tracing system, for each keyword specified by the tax officer, then there is a danger that the accesses to

the secondary index may outweigh any saving resulting from the use of the index. However, if a surname index is used to identify the single content file containing the specified surname, then only the reference file records relating to that content file need be retrieved. This will reduce the secondary index accesses to two reads per keyword.

**Table 3 Size of address reference file for various organisations and search unit sizes**

| Search unit size, blocks | Reference file size, Mbytes, using: | | |
|---|---|---|---|
| | Format 1 | Format 2 | Formats 1 and 2 |
| 6 | 894 | 5930 | 344 |
| 3 | 1054 | 11 813 | 404 |
| 2 | 1140 | 17 695 | 464 |
| 1 | 1279 | 35 342 | 592 |

### *3.4 Sizing studies*

The effect of using secondary indexes as described above has been studied using a random set of records taken from the Centre 1 Index. These have been treated as if they were specified in tracing requests. The four-character keyword stems in these records have been isolated, and the number of occurrences of the surname and each address stem in the Centre 1 Index has been ascertained from the data that was used to construct Table 1. These numbers of occurrences have then been extrapolated to the number expected in the full national tracing database.

A modelling program reads these lists of numbers of occurrences, along with various other parameters, and estimates the average numbers of search units per search and the numbers of accesses to the other files involved.

Each surname will map onto a contiguous set of search units large enough to contain the necessary number of records. The model examines the efficacy of the address keywords at reducing this number of search units to a reasonable value.

It is assumed that the tax officer specifies an average of three address words in addition to the surname, and that in general he will choose the 'best' words. Thus the model identifies and uses the three keywords from each address that occur the least number of times in the Centre 1 Index. It then estimates the final number of search units to be scanned.

Suppose

$u_s$ is the number of search units holding the given surname stem
$u_i$ is the number of search units holding the $i$th address stem (given by eqn. 2)

$p_i$ is this number as a proportion of the total number of search units $(= u_i/TU)$

then $U$, the number of search units scanned, is given by

$$U = TU \times p_s \times c.p_1 \times c.p_2 \times c.p_3$$

where $c$ is a constant used to allow for any correlation between keywords, and for the tax officer not knowing as well as the modelling program which are the best keywords. It is arbitrarily set to 1.3.

The random set of tracing enquiries has been tested against various search unit sizes. The resulting data are shown in Table 4, which indicates the average number of search units scanned in each trace after applying the secondary index optimisations, and the number of blocks to which this figure corresponds. It also breaks these down to show the percentages of the enquiries which scan more than 100, more than 10, more than 1 and just one search unit. The model can also indicate the number of accesses to the secondary index files.

**Table 4 Average enquiry sizes for different search unit sizes**

| Number of traces which scan: | Search unit size, blocks | | | | | |
|---|---|---|---|---|---|---|
| | 60 | 30 | 6 | 3 | 2 | 1 |
| > 100 search units | 4 | 8 | 7 | 6 | 4 | 1 |
| 11–100 search units | 29 | 23 | 18 | 17 | 13 | 9 |
| 2– 10 search units | 38 | 33 | 22 | 16 | 15 | 14 |
| 1 search unit | 29 | 36 | 53 | 61 | 68 | 76 |
| Average blocks scanned | 949 | 686 | 168 | 73 | 41 | 11 |
| Average units scanned | 16 | 23 | 28 | 24 | 20 | 11 |

The search unit sizes used are based upon the disc geometry: whole cylinder, half cylinder, whole track and fractions of a track.

For comparison, the average size of a search with this data if no secondary index is used is 2605 blocks, or 434 tracks. 64% of these traces are estimated to scan more than 100 tracks, and 10% more than 1000 tracks.

It is apparent from these results that the use of the secondary index is essential, and furthermore that the shortest response times correspond to the smallest search unit size. However, there are other considerations – principally the size of the address reference file (as derived above), the OCP and disc access costs of reading and manipulating the secondary indexes, the costs of fragmenting the CAFS scan into lots of smaller scans, and the overheads of updating both the content files and the secondary indexes.

Table 4 shows that a search unit size of two blocks will give the required saving in CAFS searching time. A size of one block is better, but the overheads of processing the secondary index records start to overpower the additional savings, and thus a size of two blocks is proposed.

The data generated by this modelling program has been used as input to a more sophisticated model which estimates the costs in the TPMS service of supporting the required number of tracing enquiries. It shows that a multinode Level 80 configuration will be required to handle the complete workload. But at least this is a feasible proposition, whereas without the use of secondary indexes it would not be possible to provide the CAFS power required to scan the database in the time available to meet the Inland Revenue's response-time requirements.

#### 4 Updating the tracing database

While it is moderately easy to set up and use a secondary index for a large file complex, the viability of the resulting system could be seriously jeopardised if the importance of keeping that index up to date is neglected.

There are expected to be around 192000 updates per day to the tracing database, of which half require new records to be added to the database and half require amendments to existing records.

ACO provides a number of facilities to allow updating of a file complex and the associated secondary indexes. These are used in batch mode overnight; the overheads of maintaining the indexes in step with the tracing database if the latter are subject to online updating are too large to contemplate in a system of this size.

The strategy behind the updating facilities is that if a record exists with a specific value in a field which is the subject of a secondary index, then it is mandatory that a secondary index entry links the keyword value with the search unit containing that record. But if, as a result of an update, that value is no longer contained in the record (for example because the value has been changed or the record has been deleted), then it is not essential to remove the link between the keyword and the search unit. Indeed, there may be other instances of the keyword in other records within the search unit. There is no easy way to tell. Furthermore, the effect of keeping the secondary index entry once the keyword value has been removed is that some traces will scan unnecessary search units. This may cause a small performance loss, whereas if the secondary index entry is missing, the record may well not be returned in a trace which ought to return it. Thus the updating process ensures that any keyword which appears in a new record or in a record which did not previously contain it will cause the relevant search unit to be added to the keyword's secondary index entry (unless it was already there). Periodically (for example when it is necessary to reorganise a content file to eliminate its overflow records) the secondary index will be rebuilt from scratch to remove any redundant entries.

ACO allows the content files which comprise the database to be updated either sequentially or in parallel. Because of the size of the tracing database it is necessary to use the latter approach, and thus the updates are first split into a number of intermediate update files, one per content file. These intermediate update files are then sorted and applied to the content files, performing a number of jobs in parallel to save time.

Once all the updating has been carried out, the secondary indexes can be updated using data collected during the content file updates.

Using this optimised approach, it is estimated that the daily update process will still take over four hours.

## 5 Conclusions

This paper has shown that CAFS, although it offers an enormous searching capability over reasonably large files, is not sufficient by itself for accessing very large databases in systems supporting a high access rate. The philosophy applied to the use of CAFS on VME systems to date has been that if you scan all the data available you will find the record you want. The Inland Revenue's requirements show that there are cases in which there is not sufficient time (or sufficient CAFS units) to scan all the data.

Instead it is necessary to use some intelligence to decide beforehand which parts of the database are likely to contain the required records, and direct the power of CAFS specifically to those areas. The power of CAFS to find a 'noodle in a hatrack' remains, but with so many hatracks to search it is necessary first of all to decide which fields to look in.

There are costs incurred in generating this intelligence, and indeed a multinode Level 80 configuration is likely to be required to support the application described above. However, it is estimated that considerably more nodes would be required if all the searching were done by the OCP rather than by CAFS.

The contents of the paper are derived from a proposal which ICL has made to the Inland Revenue, and much of what it contains is of necessity at present rather theoretical. Many assumptions have had to be made about the ways in which address words and surnames are distributed across the country, and a great deal of contingency will need to be built into the design of the tracing system as it evolves.

But the viability of the use of secondary indexes for this purpose is confirmed. The implementation of the ACO is complete, and by the time this paper appears it will be undergoing validation trials with the Inland Revenue.

**Acknowledgments**

The initial idea of using secondary indexing techniques to facilitate the Inland Revenue tracing system came from Alan Ward, previously of this project and now with the Software Engineering Technology Centre at Kidsgrove. The detailed design and implementation of the Advanced CAFS Option has been the responsibility of Mark Bestwick, also of this project.

Finally thanks are due to the Inland Revenue for permission to publish this paper. Without its active encouragement the pioneering techniques described above would not have got off the ground.

# Creating an end-user CAFS service

**C.E.H. Corbin**
Southern Water Authority, Brighton, Sussex

**Abstract**

The paper describes the creation and introduction of a Querymaster/CAFS service into an information-processing system that already provided numerous services under VME, and considers the impact upon the end users and the Computer-Services Department.

The paper also proposes areas where further developments are required in order to provide an improved information system that exploits CAFS.

## 1 Introduction

The introduction of CAFS into an established VME system can be viewed at its simplest as adding another tool to the VME armoury, albeit a potentially powerful one, to the many already available to the providers of an organisation's information system.

The success of introducing a new facility such as CAFS into an existing information system can be measured by the impact the new facility has upon the individual user's function within the organisation. The initial impact is related to many factors, the important factors in the context of CAFS being:

- speed and breadth of introduction
- ease of use
- amount of training required
- effect upon other services and the users of those services.

These factors and many more that relate to the implementors are considered in this paper.

The success of the introduction of the CAFS service, through a Querymaster interface, into Southern Water Authority's information-processing system is described in this paper. The paper describes not only the impact upon the users of the information system, but also the resultant behaviour patterns of users and the consequential list of new demands for facilities.

The success story is due not only to the providers of the hardware and software, i.e. ICL, but also to the approach adopted by the members of the Southern Water Authority Computer Services Department when introducing

CAFS. The paper demonstrates that what may appear to many at first sight to be a risky approach, proved not to be so and has only brought benefit to the organisation as a whole in the area of information processing.

## 2 Overview of Southern Water

Southern Water is one of ten regional water authorities established under the Water Act 1973, with responsibility for the whole water cycle within their own self-contained regions. The Water Authorities' areas are based on the concept of river-basin management. The Authorities' chairmen and the boards are appointed by the Secretary of State for the Environment.

The Southern Water Authority's geographic area of 10 552 $km^2$ covers the south east of England across the counties of Hampshire, Isle of Wight, Kent and Sussex. The resident population of the region is 3·9 million of whom 2 million are directly provided with water, and 3·7 million are connected to the main sewerage network[1]. The remaining population are either provided with water via one of the private water companies within the region or are not connected to the public sewerage network.

The number of properties connected to the water network is 0·8 million and the total length of the water mains is 11 238 km. The number of properties connected to the main sewer network is 1·6 million and the total length of the sewers is 20 085 km. The sewerage is treated in 430 works throughout the region. The main river network managed is 2731 km long.

The Authority managed in 1984/85 all of the above assets, worth £3393 million gross, with an annual capital expenditure of £65 million, and a revenue income of £159 million, with a staff of 3336 people, of whom 1354 are manual workers and 1982 are office workers. The Authority directly bills 1·8 million customers.

The Authority's information-processing systems provide applications that deal with the financial and technical aspects of its business. The size of the organisation as described results in a large amount of information that is held in computer-readable form. It is this aspect that attracted the Authority to CAFS and the exploitation thereof.

## 3 Starting line

In July 1983 the information system was a 32 Mbyte dual 2966 running under VME/B (Base 7·05) with

- 9·6 Gbytes of online filestore held on 640 Mbyte fixed discs, containing data
- 2·6 Gbytes of exchangeable file store on 200 Mbyte discs, containing the operating system and the offline databank
- 0·64 Gbytes of file store held in reserve for resilience against fixed-disc failures.

The services provided for users to access the data were

- TP enquiry service (one message pair per 0·9 s)
- TP update service (one message pair per 5 s)
- multiaccess (33 streams available)
- remote job entry (8 streams available)
- batch (3 streams available)

via 160 VDUs and over 20 remote-job-entry (RJE)/microsystems.

The data are held in index sequential files (95%) and IDMS database (5%), the distribution of index sequential file sizes being as in Table 1.

**Table 1 Distribution of file sizes**

| File size, Mbyte | Percentage of files, % |
|---|---|
| 0 – 0·1 | 17 |
| 0·1– 1·0 | 31 |
| 1·0– 10·0 | 32 |
| 10·0– 50·0 | 13 |
| 50·0–100·0 | 3 |
| 100·0–200·0 | 2 |
| 200·0–400·0 | 2 |

The data dictionary had been in use for 3 years in the computer processes and data quadrants, the version of DDS in use in July 1983 being version 650.

Querymaster had been in use since the Autumn of 1982, and available to users on a limited basis, the version of Querymaster in use at July 1983 being version 202.

The information system had been provided under VME/B since 1978.

In summary, the information-processing system was well established with many of the products that are required to exploit CAFS at that time both in place and in use, as well as the requirement to improve the search time for data held in files over 10 Mbytes. The result of this was that CAFS could be implemented and exploited quickly.

## 4 CAFS service – preparation phase

The approach to CAFS was initially cautious with the acquisition of one CAFS-ISP module. The first CAFS queries were successfully processed on 7th July 1983. The CAFS query impressed the computer staff involved not only because the query time had reduced from hours to minutes, but also with the ease that this had been accomplished; i.e. no changes to the data or

queryview. During the period from July to November numerous tests were carried out, the results of which have been well documented elsewhere[2]. As a result three further CAFS-ISP modules were ordered.

On 27th November 1983 VME release 800 (SV200), both base and superstructure, went 'live' and the first CAFS queries were run during the prime shift on files that were also accessed by TP. The CAFS search fragment size had been set to one, in order that there was no noticeable effect upon the TP service. Demonstrations were then made to a selected number of users, who immediately requested the facility be made generally available. During the period December 1983 to March 1984, CAFS accesses were allowed but under the control of the Computer Services Department. The control was exercised for the following reasons:

- dynamic switching of discs to the CAFS stream
- need to monitor the effect of the CAFS searches
- privacy.

The dual 2966, for resilience purposes, had disc drives with the dual-access feature fitted and active. With only one CAFS-ISP module, each time it became active if the disc required was not connected to it VME would dynamically switch the required disc to the CAFS stream. The effect of this dynamic switching is that the system becomes progressively unbalanced, thus affecting the other services.

The users who were initially permitted to use CAFS were those who owned the data being searched. They stipulated that access to other staff within the Authority was strictly forbidden until the Computer Services Department could guarantee secure access and the appropriate levels of privacy, together with an audit trail of when and by whom data was viewed or extracted.

To ensure that the CAFS investment was exploited as rapidly as possible the management of the Computer Services Department issued the following directives to the staff:

- 'that where ever Querymaster is in use, CAFS must be used "privacy permitting", allowing Querymaster to make the choice'.
- 'that all new application systems and new subsystems to existing application system shall be provided with appropriate views of the data available at the time of implementation'.

The directives have become part of the installation standards. The reason for the directives is to remove the decision making from implementors, who may well ponder on whether CAFS was suitable for IDMS implementations and small-index sequential files and whether or not queryviews should be made available. A further effect of these directives was that where technical problems arose these were resolved rather than bypassed.

On 14th March 1984, the three CAFS-ISP modules were accepted and brought into use, making four in total, without controls, apart from the limit on MAC VMs, provided that the user had full ownership and responsibility for the data.

### 5 Data categorisation

The Computer Services Department's management clearly indicated that Querymaster/CAFS was an end-user tool and that the Department, as providers of the 'window' or 'view' onto the data would need to be capable of providing new or amended views rapidly. Although this was technically possible, provided that the appropriate information was contained in the data dictionary, it was difficult to achieve due to the lengthy and time-consuming procedures required for obtaining authorisation for the end user to view the data.

In order to reduce the delay incurred, the Authority's Regional Management agreed upon categorisation of data, which eased the task of seeking authorisation. The categories agreed were

- statutory restricted
- authority, sensitive
- personal, sensitive
- personal, nonsensitive
- other applications.

All owners of data then indicated which records and fields within records mapped onto each of the these categories. The information derived could then be used in conjunction with the Authority's privacy subsystem.

### 6 Privacy system

The Computer Services Department in the spring of 1982 had designed a privacy subsystem for use within the TP applications. This has since been redeveloped to control nearly all the services available on the information system and replaces both the VME and Querymaster password systems; thus VME and Querymaster passwords are not used.

The privacy subsystem provides

- each person who uses the information processing system with a coherent single method of access, 'their name and their password', irrespective of the service used
- a common sign-on screen
- online maintenance
- a fine degree of control
- password security
- enforced password changing

- audit trail
- hidden checking
- common system for all applications
- ease of maintenance.

The privacy subsystem has been exploited fully since August 1984 in the Querymaster/CAFS service, to provide the levels of security and protection of data required by the Authority.

The presence of the privacy subsystem combined with data categorisation has enabled the Authority to open up the data to all who have a legitimate use for it in executing their function within the Authority, which subsequently enabled the potential of CAFS to be unleashed.

## 7 Querymaster/CAFS environment

In order to fully meet the requirement for security, and provide a user-friendly interface for the users of the Querymaster/CAFS facilities, an environment was required. The objectives to be met by the environment were

(*a*) To provide
- an efficient VM for running Querymaster
- a queryview news screen, by incorporating the standard SWA news facilities
- facilities for
  - creating and editing files of Querymaster commands
  - preparing a batch job for the appropriate application batch profile, using specified input and output files
  - routing of reports to alternative destinations
- a uniform but simple interface.

  The user is taken through a hierarchy of menus where he or she selects the options required, such as the queryviews, name of data files required where there is a choice;

  e.g. S for Sussex
  O for old year
  N for new year etc.

  the environment then assigns the files required by the requested queryviews and enters Querymaster, which is then used as normally.

(*b*) To protect the data from access other than by Querymaster.
A set of VME user names and profiles that only allowed Querymaster to be used were established. Read access is then given to the datafiles required by the queryviews available from each VME 'xxQM' username.

(*c*) To prevent the use of queryviews to which access has not been granted.

On the queryview choice template, the users are shown only those queryviews that they have been granted permission to use. This is accomplished by the SWA privacy system taking the information gained when the person logs on and then checking the personal privacy rights of that person and displaying only those queryviews that they have permission to use or from which they can then choose, should there be more than one.

(*d*) To enable session output files to be copied into the user's own filestore.

The environment not only provides a secure user-friendly interface for the users to Querymaster/CAFS, but also the advantages of standardisation and control for those responsible for providing the service. The packaging of the products to provide such a service is recommended as a prerequisite for a successful implementation. There remain a few untidy areas such as the double entry of queryview names, the users being required to identify themselves again when entering Querymaster, all of which will be cleared in a future release of Querymaster when it is expected that a packaged environment will also be provided.

## 8 User training and documentation

The Computer Services Department strongly held the view that Querymaster had been designed for end users, and as such the environment provided by SWA together with the Querymaster product had to support itself. As the information-processing system already provided comprehensive TP coverage for accessing data by key there was the added problem that most users would probably be using the service infrequently. If this was the case users might have difficulty in remembering how to exploit the facility provided to its full potential, again indicating that the service had to be self supporting. The problem areas foreseen were

- the problem of understanding the data
- knowing what data was available
- remembering how to structure queries to exploit CAFS
- the ability of potential users to structure and refine queries
- the perceived requirement for an accurate answer from every query.

All of these areas of potential concern could be tackled by training, but this would be a heavy ongoing commitment and would introduce delays when introducing new or amended queryviews. The question also needed to be asked for each point of concern, whether it was a general concern about use of the information-processing system or was particular to the new service about to be introduced. Again alternative methods for resolving these problem areas were considered and proposals for further developments put in hand. The training program then derived took all of the above points into consideration, and took the following form

- initially, all users requesting the service would be given an hour of training in the use of the service, by systems analysts and user support
- when new application systems or new subsystems to existing application systems were introduced the analysts concerned would also include the training required for the queryviews provided
- a series of halfday seminars would be presented to describe and demonstrate the potential of the facility to management, and to dispel the idea that builds up each time a new service or facility is about to be introduced that it is the magic cure for all perceived problems in the world of information processing
- CAFS specific training would be provided by regular training seminars once the number of users had stabilised
- when a user requested a new queryview from user support, training would be given upon request if the user was not familiar with the data
- a survey of users using the service would be made at periodic intervals to establish which areas required attention, and for a training program to be developed where applicable.

It can be seen from the above that an extensive training program was not entered into; the results of this decision have been noted elsewhere in this paper.

The documentation provided to users normally takes one or more of the following forms:

- Querymaster users reference card
  When a person is registered as a queryview user for the first time the Querymaster reference card is issued free of charge to the user concerned as part of the confirmation handshake that registration has been completed.
- Querymaster user manual
  Local co-ordinators of user departments purchase and hold a copy of the manual for use as a reference document. Some users purchase the manual for their own use. The ratio of users to manuals is approximately 9:1.
- Application queryview charts
  All the 35 queryviews available at July 1985 have queryview charts; these appear in the applications user manual, and are also distributed individually. A new user of a queryview would be issued automatically with a queryview chart as part of the confirmation process. Thereafter the user has to request a reissue, when invited to do so by the online news facility.
- Application reference cards
  Only one queryview has been supported with a reference card to date. It has been found that in some cases the amount of information is voluminous and a reference card is not practical.
- Application information sheets
  Four queryviews have information sheets informing the user about the use of the queryview and the data.

Again the amount of documentation issued is not large, the main documentation being that produced by ICL and the queryview charts provided by the Authority. Queryview charts in themselves, in some cases, tend to be large and in the case of IDMS-based queryviews complex. There is the added problem of producing and distributing queryview charts in a responsive manner. Automatic techniques are being considered to overcome this problem. Equivalent information to that held on the queryview chart is available by the use of the Display command within Querymaster.

In order to maintain a responsive service without incurring heavy documentation overheads each time a queryview changes, documentation has been kept to a minimum.

## 9 Categorisation of CAFS service users

The number of users using the Querymaster/CAFS service grew rapidly in the summer of 1984 to 240 employees and since then has remained constant. By July 1985 there were 242 active registered users of the Querymaster/CAFS service, which represents 12·2% of the office staff. Categorisation by function is shown in Table 2.

**Table 2 User categories**

| Function | People registered | |
|---|---|---|
| | Divisions | Headquarters |
| Administration | 31 | 0 |
| Engineering | 47 | 0 |
| Operations | 17 | 11 |
| Scientific | 8 | 0 |
| Legal | 0 | 2 |
| Customer accounts | 0 | 7 |
| Audit | 9 | 1 |
| Finance | 0 | 16 |
| Personnel | 10 | 6 |
| Computer services | 0 | 77 |
| Column totals | 122 | 120 |
| Grand total | 242 | |

The distribution of users to the number of queryviews that they have access to is shown in Table 3, which provides an indication as to the breadth of coverage provided by the Querymaster/CAFS service.

The use made of the environment under MAC by the users over the past 12 months is shown in Table 4. The usage under the batch application profiles is not included.

**Table 3 Queryviews to registered user**

| Number of queryviews | Number of users |
|---|---|
| 1 | 60 |
| 2 | 74 |
| 3 | 18 |
| 4 | 17 |
| 5 | 9 |
| 6 | 5 |
| 7 | 13 |
| 9 | 8 |
| 10 | 5 |
| 11 | 4 |
| 12 | 1 |
| 13 | 2 |
| 15 | 15 |
| 16 | 1 |
| 21 | 1 |
| 25 | 1 |
| 35 | 8 |
| Total | 242 |

**Table 4 Total use made of the environment per month**

| Month | | Cost, £ | Number of sessions | Av. length of session, hours | Total ocp, hours | Total filestore transfers, $10^6$ |
|---|---|---|---|---|---|---|
| Aug. | 85 | 20413 | 377 | 0·9 | 49 | 2·8 |
| July | 85 | 10623 | 342 | 0·6 | 25 | 1·1 |
| June | 85 | 15044 | 355 | 0·7 | 33 | 1·1 |
| May | 85 | 9351 | 381 | 0·5 | 22 | 1·2 |
| April | 85 | 9421 | 398 | 0·5 | 18 | 1·4 |
| March | 85 | 5861 | 344 | 0·5 | 14 | 0·8 |
| Feb. | 85 | 12649 | 440 | 0·7 | 30 | 1·2 |
| Jan. | 85 | 8399 | 396 | 0·7 | 22 | 1·1 |
| Dec. | 84 | 8130 | 342 | 0·6 | 20 | 1·0 |
| Nov. | 84 | 9275 | 390 | 0·6 | 24 | 1·8 |
| Oct. | 84 | 6314 | 424 | 1·0 | 14 | 1·8 |
| Sept. | 84 | 2531 | 181 | 0·5 | 6 | 0·6 |
| Mean | | 9834 | 364 | 0·7 | 23 | 1·3 |

Table 5 expresses Table 4 as a percentage of the total for all work processed per month, and provides an indication of the likely increase in load due to the introduction of a Querymaster/CAFS service.

From Tables 4 and 5 it can be seen that the extra load on the mainframe is small, as would be expected if CAFS is being exploited for nearly all queries.

Table 5 Usage expressed as a percentage of all work

| Month | | Cost, % | Total ocp, % | Total filestore transfers, % |
|---|---|---|---|---|
| Aug. | 85 | 3·7 | 3·5 | 1·6 |
| July | 85 | 2·3 | 5·0 | 0·7 |
| June | 85 | 3·3 | 6·4 | 0·8 |
| May | 85 | 2·2 | 5·0 | 1·0 |
| April | 85 | 2·2 | 4·4 | 1·0 |
| March | 85 | 1·5 | 3·6 | 0·5 |
| Feb. | 85 | 3·1 | 7·7 | 0·8 |
| Jan. | 85 | 2·3 | 6·7 | 1·0 |
| Dec. | 84 | 2·8 | 6·1 | 1·0 |
| Nov. | 84 | 2·4 | 6·1 | 1·4 |
| Oct. | 84 | 1·0 | 3·2 | 1·4 |
| Sept. | 84 | 1·0 | 2·2 | 0·6 |
| Mean | | 2·3 | 5·0 | 1·0 |

## 10 User behaviour patterns

### *10.1 Confidence building*

The two main methods by which users establish their confidence have been observed to be

- experimentation and exploration of the power of both the query language and the data. Many users have been unpleasantly surprised at the lack of integrity or consistency revealed in the data, which they previously believed was good
- gaining confidence in the results. In order to establish confidence some users have been observed to validate the results of the query using other facilities available to them such as TP, Filetab etc. This behaviour pattern has had a secondary advantage in that it has brought to light errors in the system.

The result of both categories is an initial high period of usage, not just on the new environment but also on the existing environments, which appears to last up to three months.

### *10.2 Prime users*

A prime user is defined as the user of data who also has the responsibility for maintaining that data.

In order to carry out special surveys upon data subjects, nonvital fields as viewed by the prime user have had the function of the field changed. One consequence of this is that the secondary-users' queries can be misleading if based upon the field.

### *10.3 Secondary users*

A secondary user is defined as a user who has only read access to the data.

Querymaster/CAFS has opened up the Authority's data bank to departments that previously did not have read access to this information, except for paper reports via a request mechanism to the prime user or the owner. This has shown that errors can be made in the classification, for example when the prime user is financially oriented and there is insufficient local knowledge or expertise concerning operational matters. If the secondary user has the necessary knowledge or expertise such errors can be spotted and corrected.

### *10.4 Repeated queries*

A large number of users have produced their own Querymaster macros, which are then called at regular intervals to support their activities. This tends to reduce the number of *ad hoc* queries.

## 11 User demands

The new demands that have arisen following the introduction of the service fall into two categories:

(i) demands connected with the service provided

- that the time to enter the service should be improved as the log-in phase and the navigation of the hierarchy of menus which establishes the secure environment for the user to work within can take longer than the query itself. When considering the mean session time in Table 3, this is seen to be not strictly true. The demand, however, does warrant attention
- that the number of MAC streams be increased
- that queryview reference cards be produced in all cases
- that historical and other offline data be brought online
- that regular queries that the user has developed should be provided as standard within the application concerned, under TP but still exploiting CAFS
- that a report writer be made available.

(ii) demands connected with data

- that an information system be provided that enables a potential user to establish:
  - what data is available
  - where the data is held
  - the details connected with an item of data; i.e. processing cycle field use etc.

## 12 Future directions

### *12.1 Information service about data*

In order to resolve some of the problems that have surfaced from the use of Querymaster/CAFS, work has begun on exploiting the information held within the data dictionary. The objective is to provide an interface for data users to the information held within the top quadrants of the data dictionary. Various methods of providing this interface have been considered but the one in favour currently is for the information to be extracted from the data dictionary, and structured to enable a user-friendly interface to be developed, incorporating a knowledge-based system exploiting CAFS where applicable. It is not expected that this development will be in place within the Authority for at least 12 months, as a number of activities need to proceed in parallel for this objective to be met.

### *12.2 Improved availability*

An upgrade of the dual 2966 to a superdual 2988 with an increase of 25% in online file-store capacity will have been completed by October 1985. This will permit an increase in the number of concurrent users of the Querymaster/CAFS environment.

## 13 Summary

CAFS has now been in use by users of the information-processing system for the past 21 months, and has had the major benefit of awakening the users to the state and potential of the data held. This benefit should not be underestimated, as it is doubtful whether an intensive training programme on 'the importance of data to the organisation' for users within the Authority would have been as successful.

A further benefit has been the unifying effect with regards data and its use, where the Querymaster/CAFS facility has been a catalyst in bringing together people dispersed throughout the Authority's region.

Born out of the user experience, demands are now appearing for an improved integrated information system, but it is evident that these cannot be met as easily, and for the same cost, as the acquisition of CAFS.

The paper has concentrated primarily on the use of CAFS via Querymaster, mainly as that was the interface that became available first, and further the benefit arising would be seen by the users of the system for themselves. The Authority is now beginning to exploit CAFS via the other interfaces available at SV211; the pace of exploitation, however, is dependent upon the pace of development of CAFS and the products that exploit it by ICL. The latter fact should not be lost as overall success depends on numerous organisations all working towards a common goal, at a continuous but even pace.

The paper has also only touched on a few of the many aspects of implementing Querymaster/CAFS, and has made little reference to how other developments on the information processing system can enhance the use and success of CAFS, e.g. unattended operation.

## 14 Conclusion

The environment provided has combined a number of facilities and products, all of which have enabled CAFS to be exploited successfully. The success of the implementation is also due to the large number of staff within the Authority's Computer Services Department working together towards the common objective as laid down by the management. The success has also been noted in the public domain with the Authority being awarded an office automation award for the 'Best information storage and retrieval system' at Olympia, London, in March 1985.


## Acknowledgements

The Querymaster/CAFS environment described in this paper is the result of the combined efforts and ideas of the members of the Southern Water Authority Computer Services Department and has been produced as a tribute to them.

The paper has been published with the permission of the Director of Administration of the Southern Water Authority.


## References


1 Annual report and accounts. Southern Water 1984/85.
2 'Exploiting CAFS-ISP.' Working party report, July 1984, ICLCUA CAFS Special Interest Group.
3 'Querymaster (200 level) user's reference card.' ICL TP R03785.
4 'Using Querymaster (200 level).' ICL TP R00260/00.

# Textmaster – a document-retrieval system using CAFS-ISP

**M.H. Kay**
ICL Office Business Centre, Reading, Berkshire

**Abstract**

The paper discusses the use of CAFS-ISP to support a system for filing and retrieval of word-processed office documents. The document structure is based on ODA, the proposed international standard for document interchange.

## 1 Introduction

This paper describes a project within ICL's Office Business Centre that is designing a new document-retrieval system. The system is known internally as Textmaster, and it is designed to exploit the power of the CAFS-ISP search hardware[1].

The aim of the project is to provide a system that offers full-text retrieval and is suitable for ordinary office use. Existing products such as STATUS[2] and STAIRS have achieved considerable success among research workers (including scientific, legal, and commercial applications), but they are not widely used among managers and other office workers in handling everyday paperwork.

There are a number of reasons for this. We felt that the most important were:

- the existing systems are not easy enough to use. The style of their user interfaces seems to have advanced little since the days of teletypes and punched cards. The users of modern personal computers and word processors expect better than this
- most office documents are prepared on word processors, but existing text-retrieval systems generally offer word-processing interfaces only as an afterthought
- the technique used to support free-text retrieval in existing systems – file inversion – is expensive in processing time. In an attempt to reduce the costs, new documents are usually collected together in batches and added to the database at intervals, perhaps weekly. Removing documents is often even more expensive than adding them. This makes such systems much more suitable for permanent high-value information than for rapidly changing office files

- the user interface of most systems cannot easily be tailored to the particular requirements of the subject matter and the type of user. Our experience with the ICL Committee Minutes System[3] (a free-text retrieval system designed for local authorities) suggests that by focusing on the specific requirements of a particular application it is possible to create a user interface that is far superior to that of any generalised system
- office documents are less formal than academic papers, and as a result retrieval based on content alone will not always be very successful. Often the attributes of the document – date, author, document type – will be as important as the textual content in narrowing the search. This will be especially true if the person filing the document (and wishing to retrieve it later) controls these attributes but does not control the document content
- most text-retrieval systems are free-standing: they do not allow the user to switch readily from text searching to other tasks, and they do not allow information to be extracted easily from the text-retrieval system into other systems. This is acceptable for software installed in an academic library, but not for a system offered to an office worker.

To solve these problems the Textmaster software has three main differences from existing text retrieval systems. These are:

- the CAFS-ISP search engine is used to search for documents. This enables the costs of inversion to be avoided or deferred, which means that it becomes economic to maintain a much more volatile document collection
- the software is designed from the start to deal with word-processed documents. All the information in the document is retained, including typographical details and pagination, and the document can be extracted from the text database and sent back to a word processor for further editing. Document attributes are taken from the word-processed document when available, and may be altered or supplemented at the time the document is filed.
- the software is constructed in two parts: a server containing the document-retrieval functionality and a user-interface component. The user-interface component is designed to be locally tailorable. Tailoring may adapt the interface to the type of user, or to the capabilities of the terminal being used; or it may be designed to restrict or enhance the functionality offered to the user, or to integrate it with functions available from other packages.

This paper concentrates on two aspects: the use of CAFS-ISP and the link with word processing. Although the user-interface issues have been paramount in the design process, they will not be discussed further in this paper.

## 2 Document structure and ODA

Documents have structure: they have headings, footnotes, titles, named sections, numbered paragraphs and so on. The more the retrieval software knows about the structure, the more helpful it can be in allowing enquirers to specify exactly what they want to see. If the document is divided into separate

fields, the enquirers can say which fields they are interested in, and different matching rules can be used for special fields such as dates and numbers.

The Office Document Architecture (ODA) developed by the European Computer Manufacturers' Association[4] is designed as an interchange format for word-processed documents. It allows two types of structure to be imposed on the text: a logical structure and a layout structure. The logical structure defines the partitioning of the document into sections and fields (ODA calls them composite logical objects and basic logical objects); whereas the layout structure defines the pagination of the document, with provision for features such as multiple columns. The two structures are inter-related; for example, a particular logical field may always appear in the same place on the front sheet of the document.

The structural model used by Textmaster is closely based on that of ODA. Current ICL word processors support an interchange format known as normalised document format, or NDF, which is an early implementation of a draft of the ODA standard: NDF is expected to migrate towards full ODA in due course. In the rest of this paper the term ODA is used except where the differences between ODA and NDF are significant.

Although ODA gives the potential to define a great deal of structure, it does not make it mandatory; a document may still appear as a chunk of amorphous text. No doubt documents from certain sources, such as optical character readers, will indeed take this form. In consequence of this Textmaster allows three levels of document structuring (which may be combined in various ways):

- the structure of the document is internally defined using the features of ODA. This allows Textmaster to recognise the logical components (sections and fields) of the document automatically
- the structure of the document is not defined by the ODA representation, but is implied by markers edited into the document content. This allows structured documents to be prepared on equipment that offers no support for ODA structure
- the document is regarded as being unstructured text. Logical components of the document are not distinguished. However, for filing and retrieval purposes, a number of document attributes can be defined such as author, title, keywords, and reference number; these do not form part of the document content, but are held as an auxiliary document or file card. The values of the attributes on the file card default to the values recorded in the ODA document profile if available.

The first two cases are more suitable for highly organised documentation systems: most large organisations have procedures for internal publication of formal documents, and for these the structure is fairly permanent and can be documented. For example, ICL has a 'product description' for each hardware and software product it produces: the structure and layout of these documents is centrally controlled to ensure consistency.

The third case is more suitable for casual documents: memos, informal minutes and electronic mail. In this case the structure will tend to vary from one document to another and the organisation imposes very little control over the way they are written and typed. Traditionally the person filing the documents is responsible for creating order out of this chaos, but has no authority to alter the document text: the use of the file card reflects this.

The ODA standard allows each component of a document (both logical components and layout components) to be of a defined class. For example, several sections, or several pages, may have similar attributes, and these need only be specified once. In addition the document itself belongs to a document class, characterised by its structure: an example of a document class might be the 'product description' referred to above.

This concept of document class is extremely important when considering retrieval from large collections of documents. In ODA each document is self-contained: the class definitions are included in each individual document. For retrieval, however, we need to know what structural rules apply across all documents in the same class, and for this reason we allow document classes to be centrally defined in the ICL Data Dictionary System. This also allows additional properties of each component class to be defined: for example the list of stop-words (nonsearchable terms) for a field, the names and synonyms of the fields to be used in enquiries etc.

The DDS data dictionary is used because it is a public place, where the definitions are available to any other software product that cares to use them. We chose this route because it will make it easier to exchange information between Textmaster and other Management Support products such as Querymaster and Illustrator.

The basic definition of a document class may be set up in the data dictionary by a Textmaster program which takes a specimen ODA document and analyses its structure. The current NDF implementation contains a lot of information about the layout structure, but no details of the logical structure, and so these must be added by hand.

The elements that may be defined in the data dictionary are document classes, sections (composite logical object definitions in ODA), fields (basic logical object definitions in ODA) and blocks. The block element may be used to define any of the ODA layout objects: page sets, pages, frames and blocks. Sections and blocks are recursive: a section may contain subordinate sections, and a block subordinate blocks, to any depth. An additional element, the form description, is introduced to allow a document class to have several permitted layout structures.

The structure may be very liberal, or it may impose constraints on the existence, position, and contents of each field. When documents are added to the database they will be validated against these constraints and rejected if necessary.

The structure is illustrated by the entity model shown in Fig. 1.

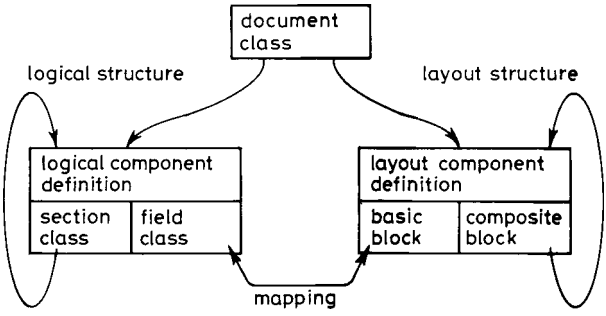


Fig. 1 Logical and layout structure of a document class

A field is a set of terms. For a text field the terms are words; fields may also be defined to contain numbers, dates, times, or arbitrary strings. The definition of a field class in the data dictionary includes rules for recognising fields of this class in the document (if this is not implicit from the ODA structure), for splitting it up into individual terms and for 'indexing' the terms. There may also be validation rules for a field class, for example a range of valid values for a date field.

### 3 Document layout

Textmaster retains the full ODA document in the text database, so that it can later be extracted and word-processed without any loss of information. There is no need to retain a separate copy at the word processor.

Retaining the full ODA document also means that documents can be displayed during an enquiry with the pagination and typography defined by the ODA layout structure (limited only by the capabilities of the enquiry terminal). They can also be printed in this form in response to an enquiry. The character repertoire used is ISO-6937[5], which caters for the needs of all Latin-based alphabets.

### 4 Retrieval capability

The retrieval capability of Textmaster corresponds broadly to that offered by established products such as STATUS and STAIRS. It includes the ability to search for any word, with stem searching, range searching and fuzzy matching, and the ability to combine search terms using the Boolean operators *and*, *or*, and *not*. The search term may be sought in any named field class; if no field is specified it may appear in any text field of the document.

A limited collocation facility is provided by the operators *with* and *without*. A *with* B means that A and B must both be present, and in the same field; A *without* B means that A must appear in a field in which B does not appear.

The main strength of the Textmaster search capability relative to other systems is in fuzzy matching. Three omnibus (or wild-card) characters are provided:

* matches one unknown character appearing at that position
? matches an unknown character, or a null string, at that position
! matches any string of characters (including the null string) at that position.

These characters can be used anywhere in a search term and in any combination. Thus, for example, STE*?EN! will match STEVEN, STEPHEN, STEPHENS, and STEVENSON.

In contrast, some of the more sophisticated features of other systems, such as proximity searching and thesaurus handling, are excluded from Textmaster, at least for the time being. We feel that these facilities will not be needed by the casual office worker.

In general all term matching ignores case and accents: thus 'COUPE' will match 'Coupé'. Similarly date and numeric fields are matched by their values rather than their representations: 1·0 will match 1·00, and 1/5/83 will match 01-05-1983. The principle is that the normalised value of a search term is compared with the normalised value of the term as it appears in the document. The normalisation rules depend on the properties of the field class, defined in the data dictionary.

Standard normalisation procedures are supplied for text fields, dates, times, and numeric fields; but the installation may use locally written procedures to normalise fields such as personal names or county names. This could be used, for example, to ensure that a search for 'Shropshire' finds a document referring to 'Salop'.

## 5 Access control

Not all users are entitled to see all documents. To regulate who may see what, some kind of access control mechanism is needed. The effect of the mechanism should be that a user who is not entitled to see a document remains unaware of its existence: there will be no evidence of it in the results of a search.

In selecting an access control model, our aims were to reflect normal office practice, to keep it simple (for the users and for the implementation) and to make it efficient at runtime.

The model we chose is borrowed from the VME operating system. It is a two-dimensional model, with a horizontal partitioning by seniority, and a vertical partitioning by department.

Each user has a seniority level between 0 and 15, and each document has a 'required seniority'. A user may only see a document if his seniority is greater than or equal to the required seniority. In addition each document belongs to a department (departments are numbered 0 to 255), and users may see only those documents belonging to departments they have access to. All users have access to department 0, but they may also have access to other departments.

A user is only allowed to see a document if both the department rule and the seniority rule are satisfied. To illustrate this, consider a user who is given seniority 10 and access to departments 0, 3 and 4. The documents he may see are indicated by the shaded areas in Fig. 2.

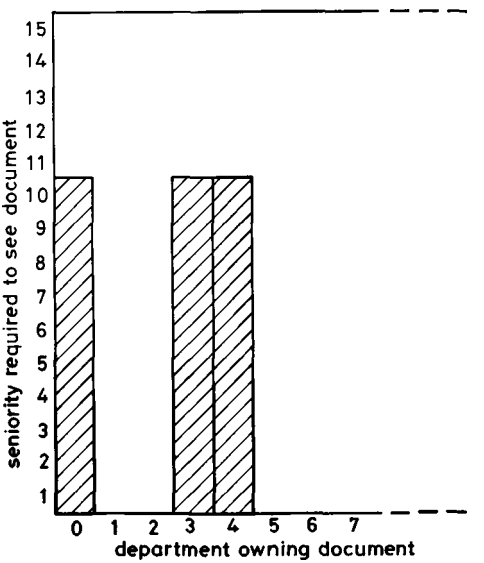


Fig. 2 Document available to a user with seniority 10 and access to departments 0, 3 and 4

Users are registered centrally by an administrator; at present we do not allow him to delegate this authority.

There are no access constraints affecting parts of a document. If a user may see a document, he may see all of it. This reflects normal office practice with paper documents. Given that documents are displayed in their original layout, suppressing part of a document would only be possible by blanking it out, which would give the game away.

## 6 Implementation using CAFS-ISP

The previous sections have discussed the functionality offered by Textmaster. This section discusses how these functions are implemented using CAFS.

We have attempted at all stages of the design to match the functionality to the perceived user requirements, rather than trying to exploit CAFS for its own sake. At times this means we have had to implement functions in software rather than using a CAFS hardware feature.

There are three main issues affecting the CAFS implementation. First, the way the documents are stored in a CAFS-searchable form. Secondly, the way in which large databases are handled when they cannot be searched using CAFS alone in a reasonable time. Thirdly, the design of the search software and the functions needed to supplement the capabilities of CAFS. These issues will be considered in turn.

### *6.1 Scan and Display Files*

Our first major decision was to keep two separate copies of the text.

The full text of the documents is kept in a *Display File* unchanged from the original ODA input except for some additional red tape to speed up display. This file is used when documents are displayed or printed, and when users ask to extract a copy of the document to their own word processors.

The text in the Display File is not searched directly, for a number of reasons. First, the text is not in the 'self-identifying format' (with each word preceded by a length byte) required by the CAFS hardware. Secondly, it would not be possible to support the function of matching normalised values, needed to support accent-blindness as well as date matching. Thirdly, the ODA form is likely to be bulky because of the amount of information retained about document layout and structure (this is even more true of the current NDF implementation). This extra information is not relevant to searching but would slow down the CAFS scan.

So a copy of the text, optimised for searching, is made in a separate file which we call the *Scan File*. This contains the normalised terms, in CAFS self-identifying format. All information about layout, punctuation, and presentation is omitted. Also, stop-words (common words such as *of* and *the*) are eliminated. We considered whether to eliminate repeated occurrences of the same word in the same field, but decided not to because the slight reduction in search time (perhaps 20%) did not appear to justify the introduction of a sort into the update process.

### *6.2 Large documents*

The next important decision was how we should handle large documents. CAFS can only apply an *and* operator if the two operands appear within some specified distance of each other; the maximum distance is limited by the maximum block size on the disc, which on the newest (FDS 2500) discs is some 23 kbytes. This represents around ten A4 pages of text. Clearly we do not want to exclude documents larger than this.

We decided to impose no limit on document size. For the time being this means that the *and* operator has to be implemented in software. A potential optimisation is to distinguish large documents from small documents on the scan file, and use CAFS to apply the *and* for documents that will fit on one record, while implementing it in software for larger documents. A more satisfactory solution would be to use the proposed file correlation unit[6] to perform the *and*. But this is not currently available.

Each record on the scan file has a fixed prefix containing the following items:

| | | |
|---|---|---|
| integer | | unique document number |
| integer | 0 .. 255 | document class code |
| integer | 0 .. 15 | seniority required |
| integer | 0 .. 255 | owning department |
| 24 bits | | department tag (see below) |
| integer | 0 .. 65 535 | field number within document |

This is followed by a variable-length part in CAFS self-identifying format, which consists of a type item, a length item and a contents item. The type item holds a numeric code identifying the Textmaster field class. (These numbers are allocated arbitrarily, so it is not possible to use the CAFS feature of masking the type code to search for several fields at once.) The length item holds the length of the value item, while the contents item is a sequence of normalised term values, each preceded by a byte containing its length. For example, the start of this paragraph would be encoded as shown in Fig. 3. Note that stop-words are omitted, and that the text is normalised to upper-case.

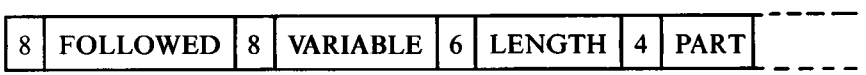


Fig. 3 Example of text in self-identifying format

A new record is always started for each new field. In addition a single field may be spread over as many records as required.

The seniority code recorded on the scan file is used to reject those documents the user may not see because he has insufficient seniority. This uses one of the CAFS key channels, with a 'less than or equal' test.

The department matching is more complex. Because the user may see documents belonging to any one of a number of departments, it is not possible to filter out the documents in a straightforward way without requiring an indefinite number of key channels. So the approach is to store a tag, which is a 24-bit value in which a single bit (department code *mod* 24) is set.

The CAFS scan matches the stored tag against a mask, which is another 24-bit value in which bits are set for each department code the user has access to.

If the department tag, when *and*'ed with the mask, equals zero, the user does not have access to the document and it can be skipped. Otherwise the document is retrieved and further examination is carried out by software. For example, if the user has access to Department 5, the CAFS scan may also return documents belonging to Department 29: these will be rejected by software.

Given that users will usually have access to a relatively small number of departments, this test will filter out most of the documents they cannot see, but uses only a single key channel.

Each record contains codes identifying both the field class and the field instance within which the terms appear. This is because there may be repeated instances of the same class of field in a document. For example a document class 'Meeting minutes' may have a field class 'Motion' which occurs a large number of times within a single document. A search may request, for example, minutes of meetings in which the words 'secondary' and 'education' appear in the same motion.

### *6.3 Large databases*

What happens when the number of documents gets so large that a CAFS search from end to end is no longer feasible?

The standard solution to this problem is coarse (or low-resolution) indexing[7]. A coarse index may be set up for any field or for all fields, and indicates for each term value in that field the range of the scan file that must be searched to find instances of that value. The advantage of a coarse index over a conventional index (which points to every instance of the value) is that it can be substantially smaller. In addition, if the presence or absence of the term in successive partitions of the scan file is indicated by a bitmap, the bitmaps for several search terms can be combined using *and*, *or* and *not* to indicate the partitions that need to be searched for a complex Boolean expression.

The alternative approach evaluated was full inversion. Here a conventional index is constructed, pointing to individual documents and possibly fields. Such an index means that the scan file version of the document can be discarded entirely, because the index contains sufficient information to identify the documents meeting the user's query.

One of the main problems with coarse indexing is in deciding a partition size. As the database grows, the number of partitions in which each term is present will also grow, and this will tend to increase the length of CAFS searches. The natural reaction to this is to reduce the partition size. Apart from the fact that this requires an expensive reorganisation of the files, it means that when the database size reaches the gigabyte range the optimum partition size becomes a single document, which means that the coarse index has degenerated into a conventional full index.

An additional drawback of coarse indexing was that it appeared to offer relatively very little performance gain on updating the database when compared with full inversion. Deleting documents (unless one is prepared simply to leave spurious references in the index, which would gradually degrade enquiry performance in a volatile database) could take considerably longer.

As a result of this analysis we decided that when databases became too big for pure end-to-end CAFS scanning we would remove documents from the scan file and fully invert them. From this point on, the scan file would be used only for recently added or transient documents, as a kind of holding area to allow the costly inversion process to be deferred until a convenient time, for example overnight. Enquiries would search the scan file using CAFS, and the inverted files by software, merging the results before presenting them to the user.

In practice it is likely that pure CAFS scanning, without any indexes, will meet the needs of very many office users. Searches of 10 000 A4 pages of text will be completed within 30 seconds or so, which will be adequate for most casual users; we know that many users of existing text-retrieval products have databases smaller than this. With a flexible facility for partitioning document collections into multiple databases, so that the user need only search those documents that are likely to be of interest, it is possible that the average database size would be even smaller. But for the really large databases that cannot be partitioned, CAFS still offers two advantages over a system relying on inversion alone: indexing can be carried out as a background process without affecting the ability to make new documents available quickly; and the index itself can be searched using CAFS: an important feature when fuzzy search terms are used.

Our analysis does not invalidate coarse indexing as a general solution; in particular it depends strongly on our previous decision to hold separate copies of the text for scanning and display.

### *6.4 Search techniques*

This section considers several features of the Textmaster searching software that may be of interest outside the pure Textmaster context.

Textmaster first analyses a search request to determine how much of the search can be delegated to CAFS. CAFS will always handle the filtering out of the correct document class, the required seniority and the first stage of department matching, as discussed above. In addition it will search for term values requested by the user. If there are more search terms than available key channels, Textmaster performs several CAFS scans and combines the results by software.

The search expression passed to CAFS includes only *or* operators. All matches against the individual terms are returned to Textmaster, which

combines the reference lists for each term using the operators actually requested in the user's search expression. On completion the final result of the search is obtained by combining the reference lists from the CAFS scan and the indexes, and removing any documents recorded in a separate list as being withdrawn.

The search terms as written by the user have to be converted into their normalised forms before passing them to CAFS. This uses the normalisation procedures for the field in question. When the user seeks a value appearing in one of several fields, the search passed to CAFS includes each field separately, because the normalisation rules could be different. When there is a large number of fields, this may lead to excessive use of key channels, so we may find it necessary to detect the common case where all text fields are normalised in the same way, and treat it specially.

Search terms including omnibus characters are matched by CAFS as far as possible. Use of the 'zero-or-one' omnibus generates two alternative strings for CAFS to search for; if it is used several times there may be more than two strings. For example if the user is searching for PHIL?IP?S, a CAFS search for PHILIPS OR PHIL*IPS OR PHILIP*S OR PHIL*IP*S is generated. If the 'zero, one, or many' omnibus is used in a search term, any characters after the first occurrence of this omnibus are ignored as far as the CAFS search is concerned; the values returned are then filtered by software. For example, if the search is for 'BR!SKY', CAFS will return all values starting with 'BR', and Textmaster software will check which of these end in 'SKY'. In the extreme case this results in a pure software scan, except of course that CAFS is still filtering on document class, field class, seniority and department.

## 7 Conclusions

Textmaster represents an attempt to apply text retrieval technology in the office. To achieve this we needed to provide good support for word-processed documents; to make the software much easier to use than its predecessors; to allow flexible updating of the document collection; and to allow the software to be adapted to local needs and to be integrated with other services available at the same terminal.

The role of CAFS has mainly been in permitting updating without incurring the cost of full inversion, except for very large document collections and then at the user's own convenience. The way in which CAFS is used is influenced by the need to retain the full document for further word processing if required.

## Acknowledgments

The Textmaster design team consists of Peter Ambrose, David Gregory, Michael Kay, John Palmer and Ed Wilson. The project manager is Adnan

Jazrawi. Jeremy Goulstone was responsible for the analysis leading to the strategy for exploiting CAFS. All contributed to the work described in this paper.

## References


1 HAWORTH, G.McC.: 'The CAFS system today and tomorrow', *ICL Tech. J.*, 1985, **4** (4), 365–392.

2 AERE Harwell: '*STATUS user manual*', Computer Science & Systems Division, 1982, ST-UM80, 3–1.

3 ICL: '*Committee Minutes System administrators manual*', 1985, Publication R50051/02.

4 European Computer Manufacturers' Association: 'Office document architecture', Sept. 1985, ECMA-101.

5 International Standards Organization: 'Coded character sets for text communication', 1982, ISO-6937.

6 BABB, E.: 'Implementing a relational database by means of specialised hardware', *ACM Trans. Database Syst.*, 1979, **4**, 1–29.

7 MALLER, V.A.J.: 'The content addressable file store – CAFS', *ICL Tech. J.*, 1979, **1** (3), 265–279.

# CAFS and text: the view from academia

**L. Burnard**
Oxford University Computing Service

**Abstract**

Text processing is not confined to the electronic office. Machine-readable text is also of growing importance for researchers in the humanities. The paper describes some experiments with, and user reaction to, CAFS as a text-processing tool. In one major project, a corpus of Shakespearian drama totalling 15 Mbytes has been made CAFS-searchable; the other project described concerns the Bodleian Library's free-text catalogue of over 1 million entries. Some difficulties inherent in the current CAFS hardware are described and software solutions proposed.

## 1 Intoduction

The scholarly application of computing to literary and linguistic research is by no means as eccentric a procedure as it once appeared. The micro has moved from the world of the electronics enthusiast to that of the colour supplement, and the word processor has ousted the typewriter from the study; it is not surprising to find that the scholar's characteristic mode of interaction with a text has become increasingly mechanised. Even in the comparative calm of Oxford University, the last decade has seen a major expansion of interest in such facilities as OCR-based data entry, high-quality computer typesetting and all the software and other resources needed to support large free-text databases. During the same period, the support offered by Oxford University Computing Service (OUCS) for work in this area has expanded correspondingly. Specialist hardware has been acquired (the Kurzweil data-entry machine and the Lasercomp phototypesetter on which a national service is offered) and specialist sotware produced, notably the Oxford Concordance Program which now has an international reputation. Specialist services such as the Oxford Text Archive are also provided[1]. CAFS is seen as complementing these facilities.

A CAFS-ISP engine was installed at OUCS in April 1984, as part of an upgrade of the service's 2988 system. The role of CAFS was initially seen as primarily to improve the performance of Querymaster when accessing large IDMS (Integrated Data Management System) and PDS (Personal Database System) databases[2]. However, with the availability of DCI in early 1985, it became possible to develop software more appropriate to our users' needs, particularly in the fields of text processing and information retrieval.

This paper describes two experimental applications of CAFS as a text-searching engine.

The first is typical of many research projects in which a large textual database is to be searched for words, patterns of words or punctuation. Without CAFS many such requirements are prohibitively expensive: the Oxford Concordance Program, for all its other merits as an indexing aid, can take as much as ten OCP minutes to produce a modest concordance to a few hundred lines of text. It is thus far too expensive to be run as a retrieval tool for any nontrivial size of datafile.

The second is typical of the many free-text retrieval applications in which a definite structure can be assigned to the data: in this case, that of a library catalogue. Here, the alternative to CAFS would be a standard information retrieval package such as Status. At OUCS, however, the only true database management system available is IDMS. Constraining textual data to fit this, although feasible, is not an appealing prospect.

The software used for both these applications is described in Section 4 below; user reactions, problems encountered and future developments being summarised in Section 5.

## 2 Beyond the concordance

As long ago as 1969[3] the pioneers of literary and linguistic computing were publishing articles pronouncing the imminent demise of the concordance as a respectable scholarly activity. When all the great works had been captured in machine-readable form, it was confidently asserted, the scholar would be at liberty to search among them, untrammelled by the dictates of any particular indexing scheme. Many great projects were undertaken (the Tresor de la Langue française, the Toronto Dictionary of Old English project, the Thesaurus Linguae Graecae[4]) whose full potential is only now beginning to be realised as the hardware and software tools to exploit these vast corpora of painstakingly assembled machine readable texts become available. Our experience seems to indicate the importance of CAFS as a means of exploiting these resources.

Our largest textual database consists of the text of several early printed editions of the works of Shakespeare. These texts were originally prepared by Trevor Howard-Hill, for use in a series of one-volume concordances to Shakespeare's plays prepared on Oxford University's KDF-9 machine and published by OUP between 1969 and 1972[5]. The original KDF-9 format tapes were converted to a rather more portable format at the National Physical Laboratory between 1978 and 1980, and the texts deposited in the Oxford Text Archive for use by other scholars. An article in the *ALLC Bulletin*[6] describes how they were subsequently re-edited and reformatted by staff working on the preparation of a new edition of Shakespeare under the general direction of Stanley Wells, to form a new Shakespearian corpus.

In 1984, a CAFS-searchable form of the texts was prepared, initially simply to evaluate the CAFS system; however, it has since proved an invaluable tool for scholars working with the texts.

The corpus currently comprises the complete *First Folio* (1623) and all substantive quarto texts, a total of about 50 plays, occupying over 15 Mbytes. Each play is regarded as a distinct text, and is divided into contexts corresponding with the printed lines of the original. Each context is tagged with codes indicating its status according to a number of criteria. These criteria include the author to whom the context has been attributed (some of the quarto plays are of multiple authorship), the text status (i.e. whether or not it is regarded as an authoritative reading of the base text), the type of copy and the compositor believed to be responsible for it as well as the play title, line number and normalised form of speaker's name.

In the case of the *First Folio*, CAFS was seen as a valuable way of substantiating compositorial hypotheses. At least 12 different compositors are believed to have worked on the *First Folio*. Whichever of these is responsible for a particular section of the folio can be of crucial importance for establishing the type of copy used, and hence its authenticity. Compositorial identity is determined largely by the occurrence of various spellings, typographical mannerisms etc. For example, passages set by compositor I contain over twice as many semicolons as passages set by any other compositor.

An important requirement of the database was thus the analysis of typographical features of the original, notably the use of punctuation and different typefaces. The markers indicating these features in the original text were processed during the conversion of the text to SIF (self-identifying format) records as follows. A 1 byte SIF identifier was used for each token and punctuation sequence defined in the text. The least significant four bits of each byte indicate the status of the associated token according to the following table:

| Bit | Set | Clear |
|---|---|---|
| 0 | punctuation | word |
| 1 | italic | roman |
| 2 | stage direction | speech |
| 3 | speech prefix | speech |

Bit 4 is always set, and the remaining bits are always clear. Thus, a word in italics will have an identifier with a bit value of 0001 0010 (hexadecimal 12), and an italic word in a speech prefix one of 0001 1010 (hexadecimal 1A). By applying a suitable mask to such identifiers, CAFS searches may be performed only for words which are in italics, or stage directions, or stage directions in italics, and so forth.

Because of the need to search punctuation, the text was converted to 'single' SIF format, using the MoveToCAFS procedure. The current version of the file is an ordinary sequential file, thus enabling addition and deletion of records using standard software.

### 3 The Bodleian pre-1920 catalogue

The Bodleian Library at Oxford, founded in 1602, is one of the six copyright or deposit libraries in Great Britain. Its long-time users have perhaps grown accustomed to the form of what remains the only reliable means of access to its riches. Three sides of the noble quadrangle taken over by the library in 1619 and still in use are taken up with the Lower Reading Room, within which are kept about 2000 stout leather-bound volumes, each measuring 42 × 30 cm and weighing more than 5 kg. These (known as the 'Guard Books') constitute the only generally accessible complete catalogue of the library's holdings. Into these volumes, generations of librarians have pasted, corrected and recorrected an estimated five million paper slips, some specially printed, some typed on antique typewriters, others handwritten in a variety of styles, ranging from Victorian copper plate to Edwardian illegible. A steady flow of new acquisitions (about 80000 each year) still keeps them busy producing new slips and incorporating them into this magnificent testimony to the durability of manual cataloguing methods.

The catalogue is arranged by alphabetical order of author or other heading, in some cases book or periodical titles, or the names of institutions, being used as the primary key. (All government papers, for example, are to be found under the heading 'Great Britain'.) Within this key, there is a well defined, if not intuitively obvious, sequence: thus for a single author, volumes of collected works are given first, followed by individual volumes in order of first publication, sometimes followed by biographical or critical works. In the case of authors such as Shakespeare or Dickens, who may have a Guard Book or two to themselves, a different method of sequencing is used which is simpler (but incompatible). Bodley's cataloguing rules are well defined, but (like the rules of many other venerable Oxford institutions) very much *sui generis*[7].

An estimated 20% of the slips in the catalogue are cross-references, some of which simply point from one form of a name to another, while others refer from the subject of a book (where this is a person or institution) to its author or editor. There is no other form of subject indexing, since the function of this catalogue is simply to enable users of the library to discover the shelf mark of a given book. This is a code enabling library staff to locate the book on the approximately 77 miles of shelving currently in use throughout the library. Only 20% of the books are stored on immediately accessible shelves, the remainder being held on 11 floors of book stacks beneath the library. A further mile or so of shelving is used up each year.

Ever since the Shackleton Report first proposed it in 1968, the library has recognised the need to convert its catalogues into machine-readable form. At

present, the Guard Books are divided into two major sections, one, containing about $1\frac{1}{4}$ million slips, describing books published before 1920 (about 2000 of which are still acquired annually), and the other for books published after that date. The post-1920 section exists at present in the Guard Books only and plans for its conversion or even replacement are still very much under discussion. This paper is concerned therefore exclusively with the pre-1920 section of the catalogue.

The process of typing and correcting the whole of the pre-1920 catalogue began in 1969 on the University's KDF-9, but moved in 1971 to the library's own mini, a Digital PDP-11/20[8]. It was decided very early on that any editorial work on the catalogue slips (for example, to make them conform to any particular proforma or structured record such as the British Library's MARC format) would prolong the task beyond what was practicable, a decision which had important repercussions when CAFS came into the picture. Only a minimal amount of consistency was imposed on the structure of each record, and it was encoded using conventional punctuation. Sophisticated parsing programs were written to check the internal consistency of the records generated, largely by the late John Jolliffe, at that time Keeper of Catalogues and subsequently Bodley's Librarian, without whose skill and enthusiasm the conversion process would never have achieved critical mass.

Typing and correction of the bulk of entries in the pre-1920 catalogue was completed in 1984. The complete catalogue will be made by merging this main file with a supplementary file of about 70 000 slips generated during revision of the main file and four other smaller files in various stages of completion, together totalling a further 150 000 slips. It should be emphasised that what is thereby created will remain what information scientists (perhaps rather disapprovingly) call 'unstructured text', because its structure is implicit in punctuation and layout rather than in tagged fields or a hierarchically organised database. The original purpose of embarking on the conversion process had been the production of a new printed or microform catalogue, not a database. Indeed, as a first step, production of an interim printed catalogue was begun in 1976, using a Diablo printer driven by the PDP-11; 73 volumes of this catalogue, covering the letters DOWN to MACZ, had been produced by 1983, when the project was suspended pending a fresh supply of Diablo printers.

After more than a decade of data capture, the Library is now starting to consider ways of making available the information held in this catalogue. The economics of producing a new printed catalogue are the subject of considerable debate, while the use of microform publishing would not be regarded as an acceptable alternative by a sizable proportion of library users. A third possibility, that of providing online access to the catalogue, now seems rather less futuristic than it might have at the start of the conversion process, for a number of reasons. One is the beginning of a long-term project to create an online union catalogue for all the other libraries within the University. The first stage of this project will involve the implementation of a common

automated cataloguing and maintenance system at three of the Faculty libraries, and its inception has already had a remarkably galvanising effect on the librarian community[9]. Another is the rapid proliferation of terminals and word processors throughout the University, which is in this respect no different from any other British University. But perhaps the most significant recent development in this context has been the arrival of CAFS at OUCS, which has enabled the Library to assess for itself the facilities that could be offered using a CAFS-searchable form of their existing datafile, at a minimal conversion cost.

As early as 1983, the Bodleian Library had been investigating the use of specialised hardware for library purposes. A Memex engine, a device in some respects quite similar to CAFS[10], was temporarily installed for one year, but for technical reasons was unable to demonstrate its full potential before being withdrawn in 1984. Following the cancellation of government funding, and in the absence of any other substantial R & D funding, the Library's investigations remain largely dependent on the willingness of manufacturers to use its unique database as a testbed for such hardware. This situation was somewhat improved when the OUCS CAFS unit became available for experimental purposes, but funding for the provision of any long-term or fullscale facility has still to be obtained, and will involve the resolution of many political issues (to say nothing of the difficulties of rewiring the current library premises).

A small sample (900 slips) of the pre-1920 catalogue file was converted to SIF in March 1985, using a simple Algol 68 program. Coincidentally, a different project began at about the same time to revise and update a specialist catalogue of early printed books held in Oxford libraries other than the Bodleian. Since this file was in the same format as the pre-1920 catalogue, it was also converted and librarians evaluating the use of CAFS given access to both catalogues. In practice, most testing and demonstrating was carried out on the latter catalogue which was rather larger (some 33 000 slips, total size 6·5 Mbytes).

Some sample Bodleian format records are shown in the Appendix. The 'minimal mark up' policy referred to above is evident. Each record begins with a unique six-character slip number, optionally followed by a numeric code defining the format of the record, and consists of a variable number of variable-length lines, separated by vertical bars. These indicate where the record may be divided for presentation purposes. The whole record is terminated by a single # character. Within each record four main structural elements may be identified: the heading under which it is catalogued, the title, the imprint (i.e. where the book was published and when) and the shelf mark (i.e. where it is currently stored). The exact sequence in which these components appear and the number of times they are repeated are both defined by the record's numeric code. Punctuation is also used to distinguish further subdivisions within these main components: thus within headings, parentheses are used to separate forenames from surnames, while the imprint

component can be subdivided into place of publication, publisher's name, date of publication and published size quite simply using only the existing punctuation and a small degree of knowledge about the valid possibilities for these fields. All cross-referencing citations were simply discarded from this initial file, but it seems probable that they would be equally easy to decompose, thus enabling extensive cross-checking of the whole file to be performed.

In the absence of CAFS, our next step, having identified the structure of the catalogue records, would presumably have been to attempt to normalise this structure and then to devise a suitable stored record format to support as many as possible of the fairly unpredictable demands that might be made of it. No doubt we would still, six months later, be arguing about the relative merits of author, title or publication date as primary access keys, about which words should feature in which indexes, and so forth. After much reinvention of wheels and reformatting of data, we would have a system optimised for certain types of access at the expense of others, probably also endowed with an albatross-like burden of indexing maintenance.

Instead, we took the line of least resistance and simply built up a single SIF record for each slip, using the slip number as its key so as to preserve the original alphabetic sequence of the file. Within the record, different SIF identifiers are used to distinguish the categories of information mentioned above, but their order of appearance is unchanged, so that on conversion from SIF to legible format, the record looks identical to what was originally typed in.

The SIF identifiers used are chosen to enable searches to be made over groups of categories as well as within individual ones, using the DCI masking facilities. For example, identifier X12 marks a word as being part of a surname, while X14 marks it as a forename. Our software can thus support searches for (say) 'Lesley' as surname, forename or either with equal efficiency. The full SIF structure is as follows:

**Identifier function**

X12 surname
X14 forenames
X16 qualifier on name (e.g. 'Saint', 'Bishop' etc.)
X18 institutional name
X1A periodical name (as author)
X1C rubric (i.e. subdivision within main heading)
X10 (mask XF0) any part of heading

X20 title
X24 parenthetic matter in title (e.g. editorial comment)

X32 place of publication
X34 date of publication

X36 size
X30 (mask XF0) any part of title

X40 shelf mark

The slip number (an arbitrary 6-byte code assigned during the cataloguing process) is used as the key for the file, to maintain the correct sequence of records within the file, although user access by this key is fairly unlikely. Multiple-format SIF is used to save space, although we were disappointed to find that the conversion routines supplied as a part of the DCI-100 product are more wasteful in this respect than they need be.

#### 4 Retrieval software

A simple interactive enquiry program, written in Algol 68, is used to search all the CAFS text files described above. This program supports all but a few of the facilities available through the DCI, including quorum conditions, counts, fuzzy matching and retrieval functions. It is currently command driven, using a language similar to that of DCI itself[11]. It is recognised that any eventual enquiry software must have a simpler user interface; at this stage however our objective was primarily to provide access to as much as possible of the DCI in a relatively straightforward and extensible way. A more user-friendly interface could be set up quite rapidly using Application Master, we are assured.

The field definition used by DCI to determine the structure of the records to be searched is held as a job space string, which is initialised to an appropriate value by the VME procedure used to invoke the search program. The user has the option to redefine this string (to suppress or rename some of the fields, or to change the interpretation of particular SIF identifiers), but this option has not as yet been exercised.

Searches are expressed using DCI's simple relational syntax and full Boolean logic, as mentioned above. Search expressions may be predefined and stored for subsequent reuse. The first time that any particular named search expression is actually evaluated, the count of hit records found (obtained automatically by the DCI count function) is also stored away, whether or not any hit records are actually returned. These counts, and the associated text defining the search, are available for display to the user on request at any time while the program is running.

Hit records are either displayed on the screen or output to a file for post processing. A number of different output formats are available, again under user control. In the case of the Bodleian file, for example, it will be possible to return hit records in a tagged format suitable for editing on a word processor, or in a standard MARC format for export to other catalogue-processing software.

The partial limit function is used to restrict the number of hit records actually returned to a user-defined maximum. We have not found any use for the other CAFS retrieval functions, nor have we as yet found any application for CAFS trailers. The former offer only numerical functions of little interest in this application, while the usability of the latter is severely restricted. The 'search units' they delimit can neither be nested within each other nor cross block boundaries.

## 5 Initial results

### *5.1 Performance*

In common with other CAFS users we have experienced hardware problems when a retrieval limit function is run in conjunction with a very high hit rate, and we have also had two minor hardware faults on the supporting disc drives. In all other respects however the performance of the CAFS unit has lived up to expectations. The determining factors in the response to any query were found to be the number of records actually output and the degree of concurrent activity on the machine (the OUCS 2988 supports a heavily used MAC service as well as a large batch load; no TP service is currently run). The resources absorbed by CAFS-aided searching were comparatively trivial, averaging less than 1 OCP second for each 10 Mbytes scanned through, irrespective of the complexity of the query. An unaided CAFS search through our biggest file, the Shakespearean corpus, takes about a minute of elapsed time during normal machine loading.

### *5.2 User reaction*

The performance benefits of CAFS for text searching are well known and need no further elaboration here, except perhaps to underline the importance of its browsing capablities. In an academic environment, where research is speculative and result-driven, the speed of CAFS is of importance because of the changes it makes possible in the characteristic mode of enquiry. The attitude to computing of the researcher for whom every search for a particular combination of words requires an overnight batch job is quite different from that of the researcher who performs such searches at an interactive terminal. Hypotheses can be tested as they are formed; searches can be determined by the results of previous searches. Our experience so far suggests that for users unaccustomed to interactive working CAFS has a qualitative effect as well as a quantitative one.

By contrast, users with experience of conventional fully indexed text retrieval systems, such as Status or Stairs, are accustomed to what might be called the 'stepwise refinement' method of text searching. For such users the fact that CAFS could not return a count of records satisfying some criterion without actually performing a search was a source of constant concern; it also appeared to have a qualitative effect on the way in which enquiries were framed.

For more naive users, the inability of CAFS to perform context-sensitive collocation searches (distinguishing the 'blind Venetian' from the 'Venetian blind', to use a time-honoured example) was very hard to understand, as was the absence of any efficient means of searching with left hand (as opposed to right hand) truncation of the search term. The first of these requirements is of such importance that the search program is now being modified to support it by software. Requests for records containing term *a* followed by term *b* (which may be another occurrence of term *a*) separated by a user-specifiable number of other terms will be converted into a CAFS search criterion returning a superset of the required records for further discrimination by the software. One early requirement of the Shakespearean database was a list of occurrences of 'that that' as opposed to 'that which', which would have been quite difficult to produce without this enhancement.

The two other areas of user-dissatisfaction most frequently reported were the inability of the CAFS hardware to support indexing functions, (i.e. to provide a list of all the different terms appearing in a particular field) and its poor support for non-Roman alphabets.

Almost the first requirement of the larger of the two Bodleian files converted was a list of all the different library codes used in the shelf mark field. The inability of CAFS to produce this is, of course, an architectural limitation of the current engine, which it is to be hoped will disappear like a bad dream with the availability of the file correlation unit. For users accustomed to conventional indexed retrieval systems, in which such a facility is obviously and inevitably provided, this restriction was both bothersome and bewildering. The software solution to the problem is simple enough, if not susceptible of improvement by CAFS.

The other major perceived weakness of the system was also particularly significant in the Bodleian catalogues, which contain large amounts of material in Greek and Hebrew, as well as all the European languages, to say nothing of various special symbols such as astrological signs, Latin abbreviations etc. Because these features have all been encoded with a view to reproducing them accurately rather than to searching for them, such simple requirements as accent-blind retrieval are almost impossible to specify. In the Bodleian files, angle brackets are used to distinguish words in non-Roman alphabets (〈G and 〉 enclose Greek words; 〈H and 〉 Hebrew ones) and accents or diacritics are represented by an escape sequence comprising an underline character followed by a digit indicating the accent to be applied to the preceding character. Thus the Greek word λόγοσ (word) might appear with an accent as 〈Glo_1gos〉 or without as 〈Glogos〉. An accent-blind search thus needs to know the potential position of any accents in order to replace each one with two 'phantom' characters. While workable, this is clearly an impractical solution to the general problem, particularly where words may have many accents. Again, this problem is partly caused by the nature of the current CAFS engine, which cannot be instructed to ignore arbitrary sequences within terms or to process escape sequences correctly.

Two software solutions exist, both of which depend on the flexibility offered by the SIF record structure. The simpler of the two involves storing an accent-free version of each accented word (either in the same file with a different SIF identifier or in a separate search file) as well as the accented, or display format, version of the word. The more complex solution is to recode the text completely, in such a way that suitable masks can be constructed to make searches accent-sensitive or blind as required.

The first solution is the more general, in that it could be extended to cater for texts using a variety of complex mark-up codes; the separation of 'searchable format' from 'display format' also has many ancillary benefits beyond the scope of this discussion. The second solution implies simpler preprocessing of the text during conversion to SIF, but is less extensible. It would require that different SIF identifers be used for words in different alphabets (which might cause problems for texts containing chemical or mathematical formulae) and that no alphabet contain more than 190 distinct characters. It would also obviously complicate conversion of recovered records for display purposes, which is already the most expensive part of the software.

## 6 Future plans

Our plans for the next year are first to continue development of our own text-retrieval software, in the current absence of any alternative, probably simplifying its user interface drastically by the provision of a screen-driven front end. A major new project about to begin involves the conversion of a large sample of ancient Greek texts (obtained from the Thesaurus Linguae Graecae archive in California) into SIF format. It is probable that we will use this database as a test bed for the accent-blind searching solution described above.

Among the many developments currently under way in the field of library automation at Oxford will be the provision of routines for converting between the current SIF format and the portable MARC format required for integration with other library catalogues. This opens the door to a number of interesting possibilities such as the extraction of specialist bibliographies, drawing on the resources of many university libraries for the primary data and local CAFS power for subsequent analysis.

As regards the Bodleian pre-1920 catalogue, we now have sufficient confidence in the capabilities of CAFS to convert a much larger sample which will be used as a test bed for the use of secondary indexing. Our current guess at the size for the entire catalogue is somewhere between 200 and 250 Mbytes. A straightforward CAFS-assisted free text search of files of this size might take as much as 20 OCP seconds which, however impressive by comparison with non-CAFS based systems, would probably still be too slow for an online system. We are therefore planning to produce a CAFS-searchable secondary index, keyed by author name and date, as a means of focusing such searches

to a smaller section of the file. One of the attractive features of CAFS-assisted indexing techniques is that the resolution of the index can be quite coarse, thus greatly reducing its maintenance overheads as well as its size[12]. The first two characters of each slip number should delimit a search to a maximum of 9000 records – less than 10 Mbytes. Since these two characters are also the most significant part of the primary key in the main file, defining the restricted search should be simple, using the DCI 'ANDTHEN' facility.

## 7 Conclusions

There can be no doubt that CAFS provides an alternative to conventional index-based text retrieval systems which is at once cheaper to implement and more efficient in performance. It also offers facilities which would be impossible to provide by other means in our environment. It is not however a panacea: some essential text-processing functions cannot easily be performed using CAFS. Nevertheless our users initial reactions have been on the whole sufficiently impressed to justify further work on overcoming these problems.

## References


1. HOCKEY, S.: 'Computing in the arts at Oxford University', in '*The computer in literary criticism: proceedings of a conference held at the University of Victoria B.C., January 1984*', University of Victoria, 1985, (in press).
2. BURNARD, L.: 'Database or knowledgebase?', in '*Towards a computer ethnology*', Senri Ethnological Studies, Osaka, 1985, (in press).
3. RABEN, J.: 'The death of the handmade concordance', *Scholarly Publ.*, 1969, **1**, 61–69. See also HOWARD-HILL, T.H.: '*Literary Concordances*', 1979, Pergamon, New York, and INGRAM, W.: 'Concordances in the seventies', in '*Computers and the humanities*', 1974, **8**, 273–277.
4 Information on the TLF is available from the Institut National de la Langue Française, CO 3310, 54014 Nancy Cedex, France. On the Old English Corpus, see FRANK, R. and CAMERON, A. (eds.): '*A plan for the dictionary of Old English*', 1973, Toronto. The TLG publishes an occasional *Newsletter*, available from T. Brunner, State University of California at Irvine, Irvine, CA 92714, USA.
5 HOWARD-HILL, T.H.: 'The Oxford old spelling concordances', *Studies in Bibliography*, 1969, **22**, 143–164.
6 Shakespeare and the Computer. *ALLC Bull.*, 1980, **8**, 72.
7 The Bodleian's '*Cataloguing rules*' were first defined by Nicholson in 1883, and subsequently revised by George Wheeler (1933, 1939).
8 JOLLIFFE, J.W.: 'Retrospective conversion: the long haul'. *Catalogue & Index*, 1985, (in press).
9 The Libraries Board Automation project was announced in the *University Gazette*, in Hilary Term, 1985.
10 HEATH, F.G. and WOYKA, J.G.: 'Memex – an information engine'. *IUCC Bull.*, 1982, **4**, 116–119. See also NEATE, G.: '*Interim report to the British Library R & D Department on Project SI/G/627*, 1985, Bodleian Library, Oxford.
11 '*Direct CAFS interface: programming guide*', 1985, ICL, Technical Publication R00421.
12 CARMICHAEL, J.W.S.: 'Application of ICL's CAFS to text storage and retrieval', in '*Protext I: proceedings of the first international conference on text processing systems*', MILLER, J.J.H. (ed.), 1984, Boole Press, Dublin.

**Appendix**

*Some sample Bodleian records*

EN46: Catcott (Alexander Stopford).|
The court of love, a vision from Chaucer.|
Oxf., 1717, 8ˆo.|
$Bliss B 153(2).#

AA11: 6 ABANO (Petrus de).|
See|
PETRUS de Abano.#

EO1011: Cawthorn (James).|
The poetical works of James Cawthorn. To which is prefixed, the life of the author. (Complete ed. of the poets of Gt. Brit., vol. 10).|
Edinb., 1974, cm.23.|
$2804 d. 37.#

EP696: 4Censor.|
The Censor [ed. by J.H. Friswell]. Vol. 1, no. 20–25.|
Lond., 1868, cm.30.|
$Per. 3974 d. 476(5).#

EP804: 73Cento novelle.|
Le ciento nouelle antike [ed. by C. Gualteruzzi].|
Bologna, 1525, 4ˆo.|
$Mason FF 462;|
–[Another ed., entitled] Libro di nouelle et di bel parlar gentile., [3 copies].|
Fiorenza, 1572, 4ˆo.|
$Douce MM 555; Mortara. 837; Toynbee 705;|
–[Another ed.].,|
Firenze, 1724, 8ˆo.|
$Toynbee 727;|
–[Another ed.]. con annotazioni di D.M.M. 2 tom.,|
Firenze, 1778,82, 8ˆo.|
$Toynbee 728;|
–6a ed.,|
Torino, 1802, cm.20.|
$Toynbee 765;|
–[Another ed.] con note dal dott. G. Ferrario., (Raccolta di novelle dall' orig. della ling. ital., vol. 1).|
Milano, 1804, cm.20.|
$4 ⟨GC⟩ 233;|
–[Another ed., entitled] Le cento novelle antiche., [3 copies].|

Milano, 1825, cm.21.|
$Douce G 221; Mortara 684; Toynbee 747.#

ER1056: 54Champion.|
The Champion, or, British Mercury [afterw.] The Champion, or The Evening advertiser, by capt. Hercules Vinegar.|
–No. 20–24, 26–38, 40–63.]
$Hope fol. 106;|
–No. 64–158.|
$Hope fol. 10;|
–No. 187.|
$Smith newsb. b. 2(5);|
–no. 196,409.|
$Don. c. 72;|
–No. 509.|
$Hope fol. 106;|
Lond., 1739542, fol. & 4^o.#

ER1071: Champion (Joseph) 1709–c.1765.|
An introduction to the counting house, or a collection of the various forms of business, as used in the merchants, or the tradesman's counting-house, performed, in their proper hands, by mess. Champion, Bland, & other emin.t masters, engr. by J. Howard.|
Lond., 1802, cm.18〈'x'〉27.|
$Don. d. 152.#

ET111: 2Chariton; of Aphrodisias.|
Di Caritone Afrodisieo de' racconti amorosi di Cherea e di Callirroe libri otto, tr. [by M.A. Giacomelli].|
[Rome], 1752, 4^o.|
$Vet. F5 d. 31.#

ET1200: 22Charles I; king of Gt. Britain.|
His majesties declaration; the reasons of his proceedings ... mentioned in severall letters, the proposals ... for a settled peace. Dec.20.|
Lond., 1647, 4^o.|
$C 14.12(35) Linc.#

EW37: Chavassius (Balthasar).|
Professio ver{ae} et orthodox{ae} fidei, additis commentariis.|
Ingolstadii, 1613, 4^o.|
$C 3.8 Linc.#

EX617: Chiari (Pietro).|
La filosofessa italiana, o sia Le avventure della marchesa N.N. scritte da lei medesima. 3 tom.|
Ven., 1753, 8ˆo.|
$Fic. 27424 f. 2–4.#

EY217: 3China; inspectorate gen. of customs.|
Special catalogue of the Chinese collection of exhibits for the International fisheries exhibition, London, 1883.|
Shanghai, 1883, cm.27.|
$[Radcl.].#

EZ1135: 3Christians.|
An inquiry into the miracle said to have been wrought in the fifth century upon some orthodox Christians, in confirmation of the doctrine of the Trinity, in a letter.|
Lond., 1730, 8ˆo.|
$8ˆo U 170(5) Th.#

# Secrets of the sky: the IRAS data at Queen Mary College

**D. Walker**
School of Mathematical Sciences, Queen Mary College, University of London

**Abstract**

The Infra-red Astronomical Satellite (IRAS), launched from California in January 1983, has made a very successful survey of the whole sky at four infra-red wavelengths and added greatly to the world's astronomical knowledge. Processing the very large amount of data produced by the mission was a major problem. The paper describes how the problem was dealt with by astronomers at Queen Mary College, using the ICL Querymaster software and the CAFS hardware search engine.

## 1 Introduction

The Infra-red Astronomical Satellite (IRAS) was launched on the 25th January 1983 from Vandenberg Air Force Base in California. The main objective of the IRAS project was to sensitively survey the whole sky at four infra-red wavelengths, these being 12, 25, 60 and 100 μm. For the following 10 months IRAS functioned almost perfectly, and every day transmitted approximately 10 million bits of data to the satellite ground station near Chilton in Oxfordshire. This data was then relayed to the Jet Propulsion Laboratory in California where it was processed to produce the final products of the IRAS mission.

The IRAS mission is arguably the most successful astronomical project ever undertaken, and over 95% of the sky was surveyed at a sensitivity almost 100 times that of previous ground-based measurements at the same wavelengths. The primary product of the IRAS mission, the IRAS Point Source Catalogue, contains approximately 250 000 sources, over 70% of which have not previously been catalogued. Before IRAS the total number of catalogued astronomical sources was less than 500 000, so IRAS has dramatically increased this number, and once astronomers have analysed the IRAS data in more detail our knowledge of the universe will be significantly increased.

Published originally in the Queen Mary College Computer Centre Newsletter; permission to reproduce here is gratefully acknowledged. (Ed.)

Indeed, already several important discoveries have been made with IRAS data. One example is the discovery of dust particles around the star Vega, which may be the precursor of a planetary system. Closer to home, IRAS has discovered bands of dust within the Solar System which are probably the debris from collisions between asteroids. Far beyond the limits of our own galaxy, IRAS has observed mysterious infra-red galaxies, the energy of which may arise from bursts of star formation, or the accretion of matter onto a black hole.

The success of the IRAS mission, however, has given rise to some problems – primarily how can such a large volume of data be accessed most efficiently? Two important IRAS mission products are the IRAS Point Source Catalogue and the Zodiacal History File, which contain approximately 50 and 100 Mbytes of information, respectively. Typically, an astronomer is interested in certain subsets of the data; for example, all the sources in the Point Source Catalogue having certain properties, or lying in a certain region of the sky. To search the Point Source Catalogue and extract a subset of sources satisfying given selection criteria would take several thousands of seconds of CPU time using a conventional Fortran program on a moderately powerful computer. Most of this time is taken up doing I/O, and clearly this method of access is not to be recommended. A conventional database-management system would take several hundred seconds of CPU time to search the whole of the IRAS Point Source Catalogue. This is somewhat more acceptable, but is far from ideal since it precludes interactive work by all but the most patient astronomers.

## 2 Data access

This searching problem is overcome by the CAFS facility at the Queen Mary College Computer Centre (QMCCC), which allows very large files to be interactively interrogated and searched at high speed. The CAFS unit may be regarded as a filter, reading large amounts of data but only passing back to the 2988 (and thence to the user) a subset of the data satisfying user-specified criteria. The important distinction between a conventional database-management system and one using CAFS is that the former selects data using software, whereas CAFS selects data by means of hardware. It is this difference which makes CAFS search times much less than those of conventional database-management systems. Using CAFS, the entire IRAS Point Source Catalogue can be searched in less than 20 s of CPU time, which is substantially faster than software-driven database-management systems. This corresponds to an elapsed time of between 1 and 2 min (assuming the 2988 system to be moderately heavily loaded), which is sufficiently short for interactive work. It should be noted that the search time is just the time to search through the file and extract the required data. Having extracted the data we often then want to output it to another file for subsequent analysis.

Thus the total time to search a file, extract data, and output it equals the search time, which depends mainly on the size of the file, and to a lesser extent on the complexity of the selection criteria, plus the time to output the extracted data, which is determined by the amount of data to be output. In extracting and outputting subsets of the IRAS Point Source Catalogue it has been found that the output time is greater than the search time if more than about 0·25 Mbytes of data is output.

At present there are two methods of accessing IRAS data on the ICL 2988 computer at QMCCC using CAFS. The first of these is by using a query language called Querymaster which allows the user to interrogate the IRAS Point Source Catalogue interactively. The second method is by means of the direct CAFS interface (DCI), which allows the user to drive the CAFS unit directly from within any high-level language program. This makes it possible, from within a Fortran program, to extract data from a file and then to analyse that data within the same program.

Querymaster has been used by a number of astronomers at QMC, and other parts of the University of London, to access the IRAS Point Source Catalogue. In some cases an astronomer is interested in a particular region of the sky. For example, Dr. Stella Harris-Law of the QMC Physics Department has been investigating a region of active star formation in our galaxy in the constellation of Taurus. Querymaster has been used to extract from the IRAS Point Source Catalogue all the sources in this region, and has then been used further to divide these sources into subgroups having different physical properties, thereby gaining insight into the range of processes occurring in the star-forming region.

Querymaster has also been used to investigate the large-scale structure of the universe. By carefully choosing the appropriate selection criteria, it has been possible to extract from the IRAS Point Source Catalogue all those sources which are galaxies. Analysis of the distribution of these galaxies in the sky allows us to investigate the degree of homogeneity of the universe. Of particular interest in this field has been the recent discovery by astronomers at QMC and the Institute of Astronomy, Cambridge, of a dipole anisotropy in the distribution of IRAS galaxies. The combined gravitational effect of these galaxies is thought to give rise to the observed motion of our galaxy relative to the microwave background radiation. This work would not have been possible without CAFS.

When using Querymaster, the user must specify the criterion for a record to be selected. In general, this consists of a number of subconditions linked by ANDs and ORs. Each of these subconditions is a condition on a single field of the data record, or on an arithmetical combination of them. For example, the condition

FLUX-3 GE 200

would mean that only records for which the field named FLUX-3 is at least

200 would be extracted. In the context of the IRAS Point Source Catalogue this condition extracts sources for which the flux density at 60 μm is greater than or equal to 2 Janskys. The condition:

FLUX-3/100 GE 2

has exactly the same effect.

A somewhat different type of subcondition is

FLUX-4/FLUX-3 LT 1

This places a restriction on the value of the ratio of the two fields FLUX-4 and FLUX-3, and in the context of the IRAS Point Source Catalogue requires the flux density at 100 μm to be less than that at 60 μm for the source to be extracted.

Subsets of sources extracted using Querymaster may be listed to a VDU screen, printed on a line printer or written to an output file on disc, and may be sorted on any field. In some cases, however, we only want to know how many sources satisfy a certain selection criterion and are not interested in the details of each source. For example, we might want to know how many galaxies lie in a certain region of the sky. Querymaster allows the user to count the number of sources satisfying the selection criterion without listing each one.

Querymaster is most suited to extracting moderately sized (about 1000 sources) subsets of the IRAS Point Source Database, since, as mentioned earlier, the extraction and output of large subsets is limited by I/O time. The direct CAFs interface, however, obviates the need to output subsets of sources by allowing the user to extract the desired sources and process them within the same user-written program. This allows specialised utility programs to be written for processing subsets of the IRAS data.

## 3 Applications

One application of DCI has been to access the IRAS Zodiacal History File (ZHF). This 100 Mbyte file gives the emission at each of the four IRAS wavelengths averaged over $\frac{1}{2}$ degree square pixels, and thus gives the background emission due to extended structure on scales greater than about 1 degree of arc. The contents of the ZHF are currently being used at QMC to look for isotropic background emission at 100 μm, which might possibly have been caused by bursts of star formation in distant galaxies. A DCI program has been written to draw contour plots of the ZHF background emission for user-specified regions of the sky. This is useful in looking for large-scale extended structures, such as shells of dust created by supernova explosions.

Another application of DCI has been to produce plots of regions of the sky showing the IRAS point sources in that region. The example sky plot of Fig. 1 shows the IRAS sources near the Galactic plane.

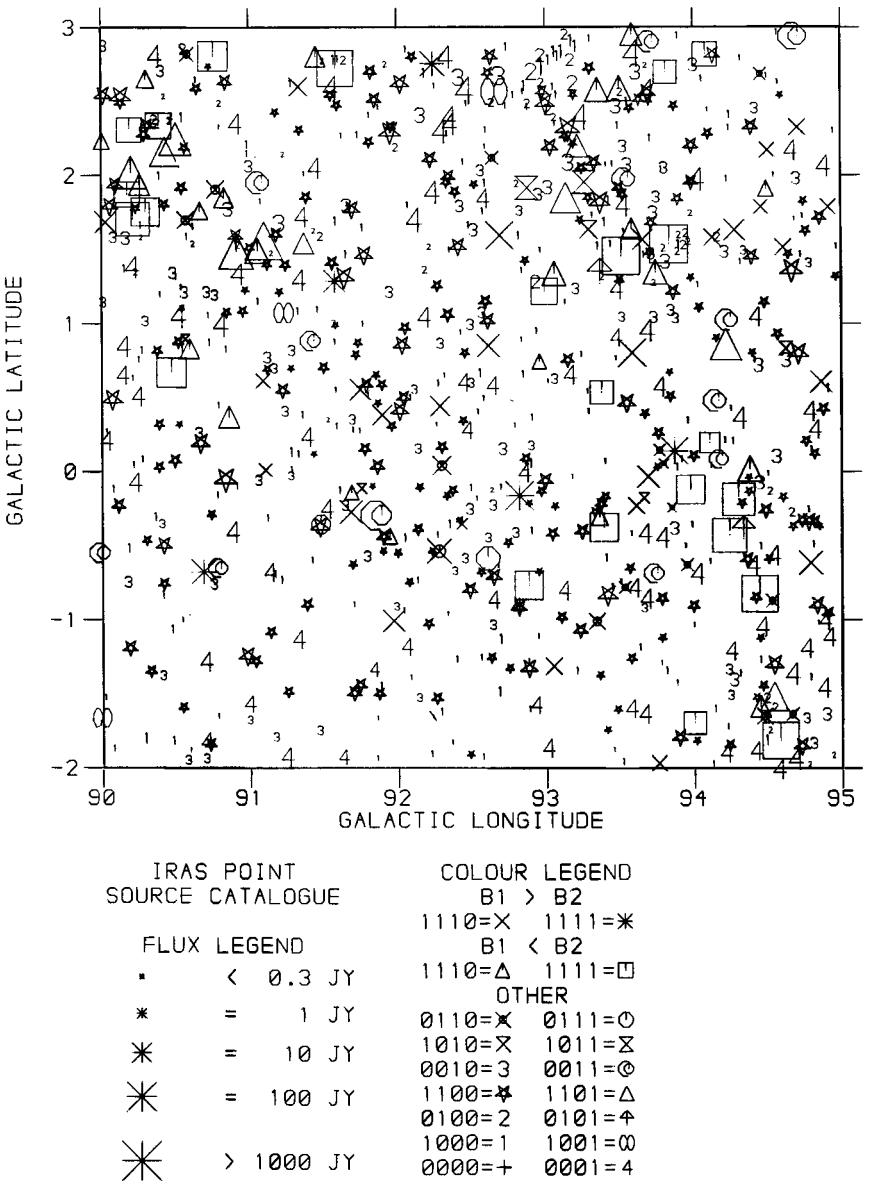


Fig. 1 Plot of IRAS sources near the galactic plane

The size of each symbol gives the maximum flux density over the four IRAS wavelengths and the colour shows at which wavelength this maximum occurs. Thus a source represented by a black (red, green, blue) symbol has a maximum flux density at 12 μm (25, 60, 100 μm). The type of symbol indicates at which wavelengths the source was observed. Each symbol is related to a four-digit code given by the legend on the left of the figure which should be interpreted as follows. A '1' in the *j*th largest position from the left means that the source was observed at the *j*th largest of the IRAS wavelengths. For example, a source with code number 1100 was observed at 12 and 25 μm, but not at 60 or 100 μm. Similarly, 0001 means that the source was only observed at the longest IRAS wavelength of 100 μm.

Sky plots such as these are very useful for gaining a quick impression of the number and type of IRAS point sources in a given region of sky, and are much easier to look at and interpret than pages of line-printer output.

The data collected during the IRAS mission seems certain to have a major effect on our knowledge of the universe around us. However, the full benefit of the outstanding success of IRAS will only be reaped if the IRAS data can be efficiently accessed and analysed. The use of CAFS at QMC has allowed astronomers within the University of London to begin to unlock the secrets contained within the IRAS data. Much of this work would have been impossible without the use of CAFS, and it is expected that CAFS will continue to play an important part in the analysis of the IRAS data.

# CAFS file-correlation unit

**E. Babb**
ICL System Strategy Centre, Technical Directorate, Bracknell, Berkshire

**Abstract**

The paper describes special hardware designed for and built into the original prototype Mk2 CAFS to assist the relational operations of join and projection. This hardware, now called the file-correlation unit, consisted of a cluster of bit maps which were addressed by data values found in the records of a file. Assistance in relational operations was provided because the bit maps were able to encode intermediate results, such as part numbers, for communication between and within files.

The hardware was built by a team lead by the late R.W. Mitchell as part of the CAFS project; subsequently a team lead by L.E. Crockford implemented the complex software to drive the unit.

## 1 Introduction

When the original prototype of CAFS was designed and built in 1974, various database applications were mounted on it with a view to determining its performance and assessing its limitations. It was soon found that there were two problems. First, it was difficult to communicate information from one record to another, for example to obtain the parts made by suppliers X and Y where X, Y were in separate records, maybe also in separate files. Secondly, it was found that the CAFS count and total operations could give misleading results: CAFS counted the number of suppliers in the database, but what the user really wanted was the number of *different* suppliers.

Various solutions to these problems were proposed, including a very complicated record structure; but the simplest solution appeared to be a bit map store as described in this paper. By making it an integral part of CAFS it could be driven directly by data coming off the disc heads. Such a store has been given the name of file-correlation unit (FCU); it assists queries involving many files by storing intermediate results, such as lists of part numbers, as a bit pattern. Thus these part numbers can be transmitted from file A to file B, and the order numbers transmitted from file B to file C. In another use it can

This paper is essentially an abridged version of a paper that appeared in the *ACM Transactions on Database Systems* (TODS) in March 1979. Permission to use the original material is gratefully acknowledged.

remove repetition in retrieved data by storing the first occurrence of, say, a delivery date; subsequent occurrences are not retrieved because they are now detected in the FCU store.

A detailed account of the FCU, with information on performance and error rates and a mathematical treatment of some of the processes involved, was given by the present author in *ACM Transactions on Database Systems*, March 1979 (later referred to as the TODS paper[1]). The remainder of the present paper is essentially an abridgement of the TODS paper, with some slight modifications. Details of how to perform join and projection operations are given, followed by a description of the design and operation of the single bit map FCU. Because this latter involves compiling a special index into the file, to address the bit map directly, its use is limited to certain joins and projections; therefore an enhancement, called the hashed bit map FCU, addressed by data values in the file, was developed so as to assist arbitrary joins and projections. This also is described here, but not the mathematical details of the hashing process; for these and for the details of performance and error rates the reader is referred to the TODS paper.

## 2 CAFS hardware

Other papers in this issue of the *ICL Technical Journal* deal with a variety of aspects of the Content Addressable File Store, CAFS – history, general principles, performance, current hardware and applications. This paper is concerned with particular features of the prototype. This used 60 Mbyte discs with an instantaneous scanning rate of approximately 4 Mbytes/s, giving typically about a 2 Mbytes/s rate visible to the user; and this search performance was available to many tasks in parallel.

Fig. 1 is based on a diagram by Mitchell[2,3] and gives the outline architecture of the content addressable file store (CAFS). The selector (18) is used to set up the key registers (3–5) with just the key value pairs taken from the selector (18). The tuple (2) from a file on disc pack (1) enters each key register and sets a latch if the value in a tuple is 'greater than,' 'equal to', or 'less than' the value in the key register. For example, unit (5) has its '>' latch set because 4p in the tuple is greater than 3p in the key register. The outputs of the key registers then go to latch comparators (6–8). These contain the selector condition associated with the key registers on the left and originally derived from the selector (18). As can be seen, only if the condition such as in (8) is obeyed will the output line be set to 1. The outputs of the comparators (9–11) then go the search evaluation unit (12) where they form part of a logical expression again derived from the selector. If this expression is true then route (13) is set to 1 and the tuple (2) is considered to be a hit. The retrieval unit (14) then allows the components (17) of a hit record to pass on the host computer. The search evaluation unit (12) can handle nested Boolean expressions and threshold functions. Moreover, the search hardware can support concurrent tasks on the multiplexed output of up to 12 disc channels. This contrasts with the logic per track designs of Parker[4], RAP[5], CASSM[6–9] and RARES[10], which generally have only limited selection capability per scan but possibly greater parallelism.

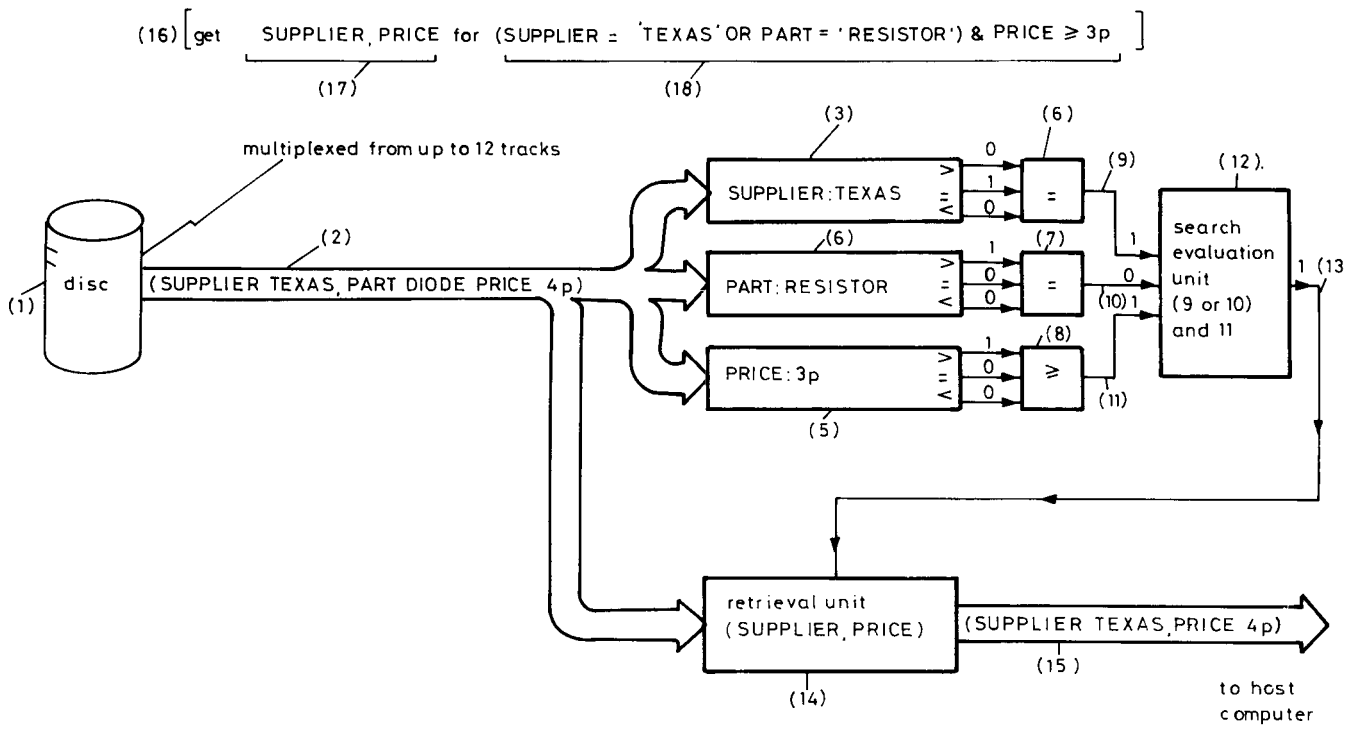


Fig. 1 Basic CAFS architecture

This paper considers two extended versions of the CAFS hardware. The first uses single-bit directly addressed bit maps to hold intermediate results and the second uses multiple bit maps and hashed addressing. The former method of holding intermediate results in a single bit map bears a strong resemblance to the CASSM system[6] (3) and is similar to an earlier idea patented by the author in 1974[11]. A more detailed comparison with CASSM is given later in the paper.

## 3 Relational operations using a single bit map

This section shows how a single bit map helps to perform relational joins and projections; it may be addressed by using either the key itself or a compact representation of the key.

### *3.1 Queries involving the join of relations*

Performing a join is best explained by an example. Consider a query involving the join of two relations *SP* and *PC* illustrated in Fig. 2, linked by their shared 'part' field. Suppose a list of parts that could be supplied by Mullard to ICL is required. One method of answering the query is to physically join *SP* and *PC* via the part link to give the relation *SPC* illustrated in Fig. 3, from which it is clear that only 'transistor' is supplied by Mullard to ICL.

*SP*

| Supplier | Part | Price |
|---|---|---|
| Texas | transistor | 9p |
| Mullard | transistor | 9p |
| Texas | diode | 4p |
| Ferranti | diode | 4p |
| Erie | resistor | 3p |

*PC*

| Part | Customer |
|---|---|
| transistor | ICL |
| transistor | GEC |
| diode | Plessey |
| diode | RRE |
| resistor | ICL |

Fig. 2 Joining relations *SP* and *PC* on a common part

| *SPC* | Supplier | Part | Price | Customer |
|---|---|---|---|---|
| 1 | Texas | transistor | 9p | ICL |
| 2 | Texas | transistor | 9p | GEC |
| 3 | Mullard | transistor | 9p | ICL |
| 4 | Mullard | transistor | 9p | GEC |
| 5 | Texas | diode | 4p | Plessey |
| 6 | Texas | diode | 4p | RRE |
| 7 | Ferranti | diode | 4p | Plessey |
| 8 | Ferranti | diode | 4p | RRE |
| 9 | Erie | resistor | 3p | ICL |

Fig. 3 The join of relations *SP* and *PC* on a common part

To answer this query in the above manner on a large relation would be possible and simple, but impractical because it involves a lengthy join of *SP* and *PC* and then a selection. The usual technique is to perform this lengthy join as late as possible, by performing selection on the two relations separately and then joining the resulting subsets of *PC* and *SP*. To further improve selection on a given file, references between relations (PART in this case) are remembered from relation 1 and then used as part of the selector on relation 2.

Lines (1)–(3) below are a procedure for performing a join followed by selection using CAFS. This procedure corresponds to the earlier query: 'Get the list of parts that could be supplied by Mullard to ICL.' Line (3) gives the answer in list *P*.

(1) clear the bit map
(2) store in the bit map all the parts associated with the supplier Mullard in the *SP* relation (the bit map now contains 'transistor').
(3) retrieve into a list *P* all the parts that have been stored in the bit map and are supplied to customer ICL in the *PC* relation (list *P* now contains 'transistor').

Each line above is a CAFS command written in English. However, what is not made clear is the physical form of the bit map, and it is necessary to describe this before going into more detail about the procedure.

Consider the list of parts (transistor, diode, resistor). The object of the bit map is to codify subsets of this list in a compact manner, which is done by associating an index with each member of the list of parts. Thus transistor is labelled 1, diode 2, etc., to give a list of part-index pairs [(transistor, 1) (diode, 2) (resistor, 3)]. Any subset of this original list can now be uniquely coded by a three-element bit map with each bit position corresponding to a different part as indicated by the part-index pairs; a bit set to 1 would indicate a part in the subset. Thus the pattern (1, 0, 1) would codify indexes (1, 3) which label the (transistor, resistor) sublist.

The bit map is addressed by indexes stored in the database as extra, precompiled fields called 'coupling indexes.' Fig. 4 shows a new version of the relations *SP* and *PC* in Fig. 2 with a coupling index $I_{PART}$. Indexes are assigned to different parts using the numbers 1, 2 and 3.

| *SP* | Supplier | Part | Price | $I_{PART}$ |
|---|---|---|---|---|
| 1 | Texas | transistor | 9p | 1 |
| 2 | Mullard | transistor | 9p | 1 |
| 3 | Texas | diode | 4p | 2 |
| 4 | Ferranti | diode | 4p | 2 |
| 5 | Erie | resistor | 3p | 3 |

| *PC* | $I_{PART}$ | Part | Customer |
|---|---|---|---|
| 1 | 1 | transistor | ICL |
| 2 | 1 | transistor | GEC |
| 3 | 2 | diode | Plessey |
| 4 | 2 | diode | RRE |
| 5 | 3 | resistor | ICL |

Fig. 4 Relations *SP* and *PC* with special coupling index $I_{PART}$

CAFS contains one or more bit maps $M1, M2, \ldots$ addressed by coupling indexes. For each tuple in a relation the addressed element can either be set to 1 or 0 or be tested for being 1 or 0. Thus for some tuple containing $I = 3$ the following occurs:

- set $M(I) := 1$ (leaving $M$ with (0, 0, 1) and adding a member to the coded set)
- perform some action if $M(I) = 1$ (i.e. test membership in the coded set).

It is now appropriate to introduce the CAFS architecture with bit maps.

Fig. 5 can be used to describe essentially how this enhanced CAFS operates. A query from the terminal (34) causes a task to be sent from the host computer to all the various units shown. Unit (33) selects a disc drive (1) and allocates channels. Tuples (2) are then sent to the key registers (3–8), bit address filter (21), and retrieval unit (14). The bit address filter extracts the index value $I$ from a coupling index such as $I_{PART}$. This address $I$ is then passed to the store where it can be used in either or both of the following two ways:

- the contents of the address $I$ can be *read* from the bit store (23) and sent along route (25) for use by the search evaluation unit (12). Thus a test such as $M(I) = 1$ can be included anywhere and any reasonable number of times in the selector
- operation of the bit map (23) can be delayed until unit 12 has operated on a complete tuple. If there is a hit in route (13) then a '1' or '0' is *written* into address $I$ in map (23).

Once a bit has been set in route (13) then the retrieval unit (14) allows requested components of the tuple through to the host computer (32), and then after suitable processing to the terminal (34). In general, there are a number of bit maps (30) each independently addressed and each containing at least 64 K bits of memory.

In contrast with CAFS, CASSM[6] uses one physical bit map per track, so the implementation of one virtual bit map, addressable by all tracks, requires the address of a bit to be passed from one logic unit to another until it arrives at the appropriate physical bit map. Each transfer requires a hardware adder/subtractor to operate on this address. There are additional timing problems which sometimes mean that more than one scan is needed to set or test the virtual bit map. This probably means that the CASSM system could not perform projection as described in Section 3.2. Although the process of joining two files is not actually described in Reference 6, there seems no reason to believe that CASSM would not, in fact, be able to perform this function.

Returning to the original query, it is now possible to give an explicit description of how the query: 'Get the list of parts that could be supplied by

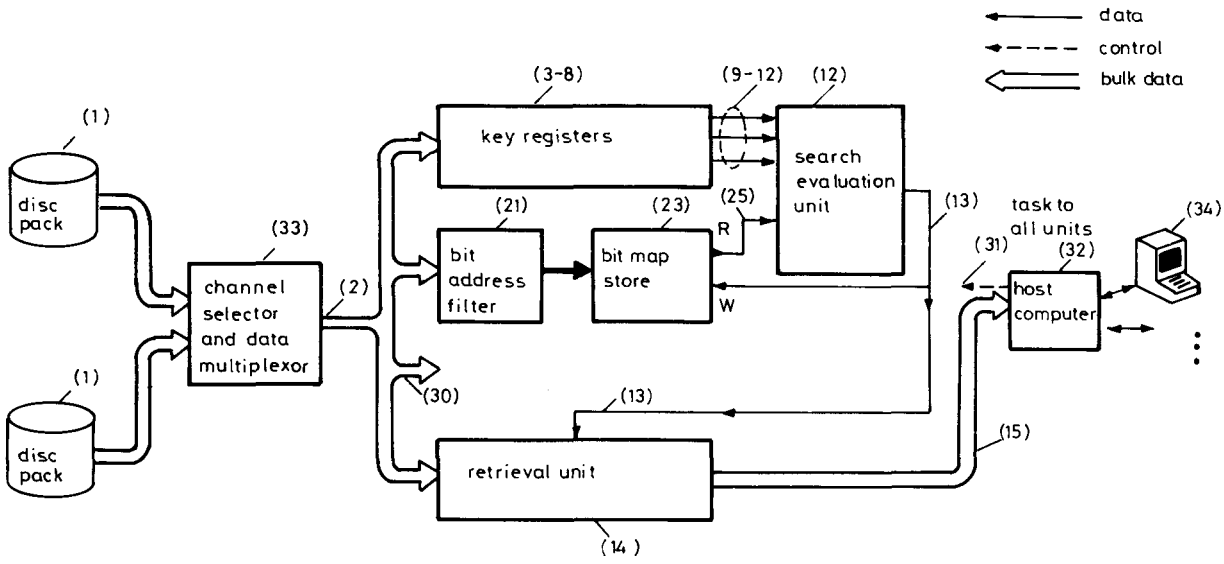


Fig. 5 CAFS architecture with single bit map

Mullard to ICL' can be answered. As before, each line below is a distinct CAFS command operating on the relations in Fig. 4.

(1) clear bit map $M$
(2) for each tuple (SUPPLIER, PART, PRICE, $I_{PART}$) in relation $SP$, if SUPPLIER = 'MULLARD' then set $M(I_{PART}) := 1$
(3) for each tuple ($I_{PART}$, PART, CUSTOMER) in relation $PC$, if $M(I_{PART}) = 1$ and if CUSTOMER = 'ICL' then put PART into list $P$.

Line (1) clears $M$ and hence causes it to represent an empty set of parts. Line (2) stores the parts supplied by Mullard using $I_{PART}$ and sets the first bit of $M$ to 1. The final bit pattern in $M$ is (1, 0, 0) and represents a 'transistor.' Line (3) puts the actual part value into a list called $P$ if element $I_{PART}$ of $M$ is 1. Clearly, from the $PC$ relations Fig. 4, only tuples 1 and 2 make $M(I_{PART}) = 1$. But the customer must be only ICL and so only the value 'transistor' from tuple 1 is actually retrieved.

The bit map enables the volume of data returned for final joining to be minimised by storing coupling indexes from one relation to another in a special code. The bit map can be used to follow coupling indexes through a chain of relations just as one would with physical pointers. At a more practical level only 64 K bits of storage are needed to store up to 64 K possible parts or whatever the coupling indexes might be.

### *3.2 Queries involving the projection of a relation*

As mentioned earlier, projection consists of removing some of the fields of a relation and then removing any repetition in the remaining data. A projection of Fig. 6 is shown in Fig. 7 where the supplier field of the $SP$ relation of Fig. 6 has been removed to give $P$, the part-price relation.

$SP$

| Supplier | Part | Price |
|---|---|---|
| Texas | transistor | 9p |
| Mullard | transistor | 9p |
| Texas | diode | 4p |
| Ferranti | diode | 4p |
| Erie | resistor | 3p |

Fig. 6 Relationship $SP$ between suppliers and parts

$P$

| Part | Price |
|---|---|
| transistor | 9p |
| diode | 4p |
| resistor | 3p |

Fig. 7 Projection of $SP$ to the PART and PRICE fields

The conventional method of projection can be slow and expensive and would consist of fetching whole records into a host computer, picking out the required fields, and then performing an interlist comparison or its conceptual equivalent to remove repetitions.

The CAFS approach, using the bit map, consists of two steps: (The example corresponds to the command: 'Get parts and their prices.')

(1) clear the bit map
(2) store each part and price in the bit map and in list $P$ if and only if these parts and prices have not already been stored in the bit map.

As before, each line is a CAFS command. The function of the bit map in line (2) is to remember what has been placed in $P$ already and not to retrieve it again, by always checking the bit map.

The explicit CAFS store is a bit map. As one might expect, the part-price pairs must be associated with a precompiled index $I_{PART,PRICE}$ illustrated in Fig. 8. The two lines above become the following commands:

| $SP$ | Supplier | Part | Price | $I_{PART,PRICE}$ |
|---|---|---|---|---|
| | Texas | transistor | 9p | 1 |
| | Mullard | transistor | 9p | 1 |
| | Texas | diode | 4p | 2 |
| | Ferranti | diode | 4p | 2 |
| | Erie | resistor | 3p | 3 |

Fig. 8 Relation $SP$ with index to help projection

(1) clear bit map $M$
(2) for each tuple (SUPPLIER, PART, PRICE, $I_{PART,PRICE}$) in relation $SP$, if $M(I_{PART,PRICE}) = 0$ then:
    (a) put (PART, PRICE) into list $P$
    (b) set $M(I_{PART,PRICE}) = 1$.

Observe tuple 1 in Fig. 8. Line (2) will put (transistor, 9p) into $P$ and set $M$ to contain (1, 0, 0). Tuple 2 contains $I_{PART,PRICE} = 1$, and since the test $M(1) = 0$ fails, the repetition of PART, PRICE is not put into $P$. Similar events occur for all the other tuples to give a true set in $P$.

For simplicity, no selection operation has been included in this example. If a selection operation had been used, then the relation would appear to contain fewer tuples. Practical sized bit maps of 64 K bits allow one command to correctly project to a new relation containing 64 K tuples.

## 4 Relational operations using the hashed bit map store

In this section a method of addressing the bit maps using hashing techniques is described. The advantages of this approach when compared with the former methods are:

- the key value can be outside the store limits and the compact representaton is not needed
- the key value can be formed from several arbitrary fields.

### *4.1 Architecture of the hashed bit map store*

The hashed bit map store consists of one or more bit maps arranged as shown in Fig. 9. No coupling indexes are required since the store is effectively addressed by any *concatenation of field values*. Thus a tuple (2) containing values (TEXAS, TRANSISTOR, 9p) is converted into the key $K$ by the address filter (21) to give, say, (TEXASTRANSISTOR) as a single bit string. The key $K$ is then hashed to addresses $J1, J2, J3$ by the hash coders (22). One hashing function discussed by Knuth[12] is division modulo $b$ which generates an address between 1 and $b$ using ($K$ mod $b$) + 1. Different hashings can be obtained by sorting the bits $b_0, b_1, \ldots, b_n$ in $K$ into a new order $b_{i_1} b_{i_2} b_{i_3}$ to give $K_i'$ and using division modulo $b$ to give $J_i$. It is very important that each hash coder (22) behaves quite differently so that $J1, J2$ and $J3$ are statistically independent. As with the single bit map, this store can be used in either or both of the two ways listed below.

- the contents of addresses $J1$, $J2$, $J3$ are each read using (23) and then 'Anded' together by unit (24). If they all contain 1 then a 1 is passed along route (25) to the search evaluation unit (12) for inclusion anywhere in a selector. The effect of 'Anding' will be explained later in this description
- when a tuple has been evaluated by unit (12), a '1' or '0' is written into all the stores at addresses $J1$, $J2$, $J3$ if route (13) contains a hit.

More formally, if $h_i(K)$ is a hashing function then the store is read from or written to as follows:

- to write $K$ to the store:
  $M1(h_1(K)) := 1$, $M2(h_2(K)) := 1$, $M3(h_3(K)) := 1$
- to read $K$ from the store:
  $M1(h_1(K)) = 1$ & $M2(h_2(K)) = 1$ & $M3(h_3(K)) = 1$.

Consider just one map $M1$. Now hashing functions are many-to-one mappings from $K$ to $h(K)$. This means that two keys $K_i$ and $K_j$ can share the same address $J1$ since $J1 = h(K_i) = h(K_j)$ for some $i, j$; such an occurrence is known as a collision. Now this phenomenon can be used to explain how the store can appear to contain spurious members. Suppose $K_i$ is stored in the memory using $M(h(K_i)) := 1$. Now if $M(h(K_j)) = 1$ is tested, and if $h(K_i) = h(K_j)$, then $K_j$ will appear to be in the memory. Thus if keys $S_h = (K_1, K_2,$

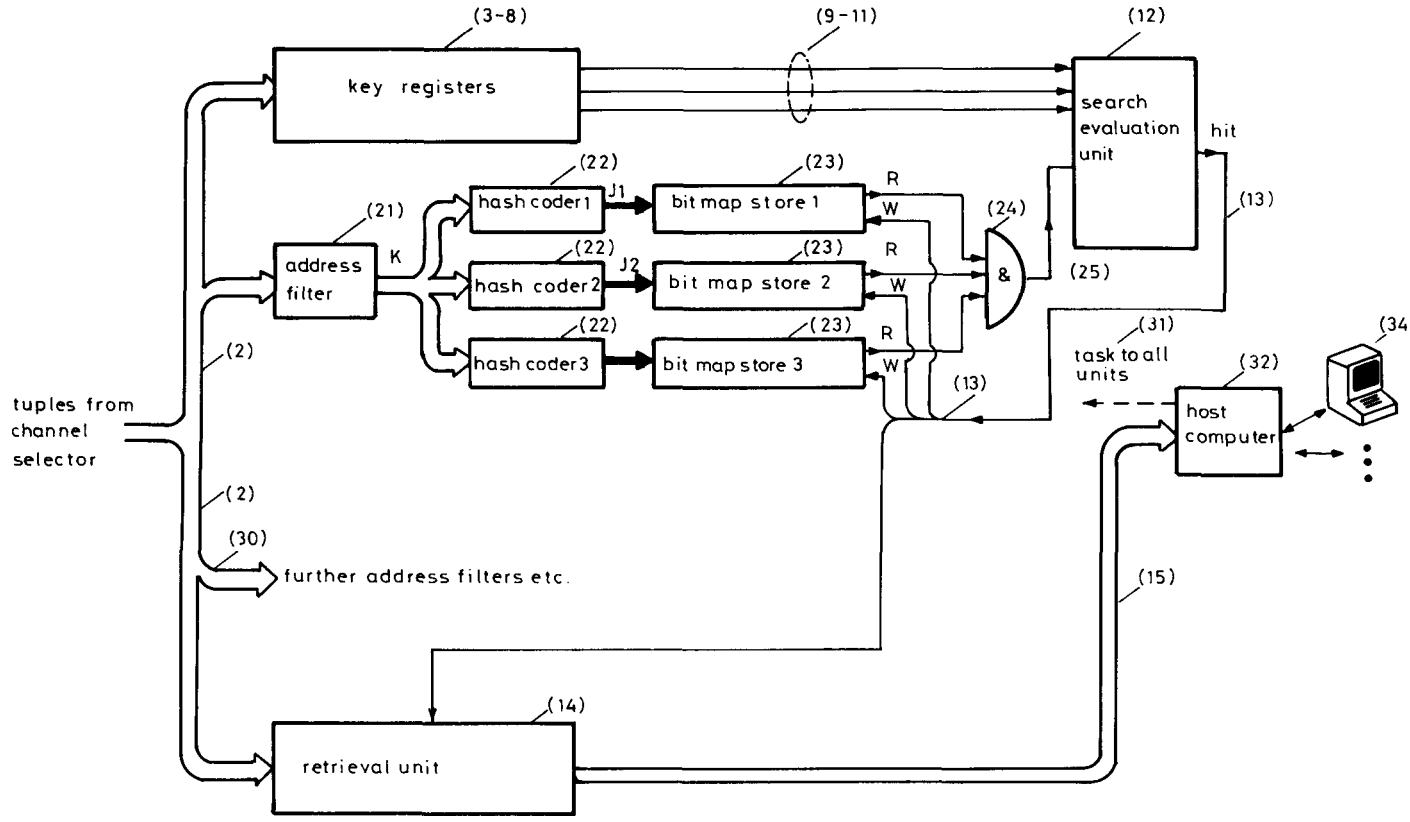


Fig. 9 CAFS architecture with hashed bit maps

$K_3 \ldots K_h$) are written into a hashed bit map during one CAFS scan, then a subsequent test scan over a wider set of keys $S_r$ will find an output set $S_k = (K_1 \ldots K_h, K_{h+1} \ldots K_k)$ where $K_{h+1}$ etc. are the result of a collision with a member of $S_h$. The crucial property of the hashed bit map is that $S_k$ *is a superset of* $S_h$.

The purpose of the 'Anding' unit (24) in Fig. 9 is to reduce the proportion of spurious members. Suppose the set $S_h$ is stored in two bit arrays $M1$ and $M2$. Now $M1$ and $M2$, when tested with the set $S_r (S_r \supseteq S_h$ by definition), will appear to contain the sets $S_{k_1}$ and $S_{k_2}$ of keys as shown in Fig. 10.

As mentioned earlier, $h_1(K)$ and $h_2(K)$ are statistically independent. Thus it is very unlikely that spurious members in $S_{k_1}$ and $S_{k_2}$ will overlap. Thus a good approximation to $S_h$ will be $S_{k_1} \cap S_{k_2}$, especially if $|S_h| \ll |S_r|$. This, therefore, is the reason for 'Anding' in unit (24), since this checks that $K$ is in both $M1$ and $M2$. More specifically,

$$M1(h_1(K)) = 1 \;\&\; M2(h_2(K)) = 1$$

In general many stores may with advantage be 'Anded' together: details of the resulting performance are considered in the TODS paper.[1] Also there can be many hashed bit map stores connected at (30) in Fig. 9.

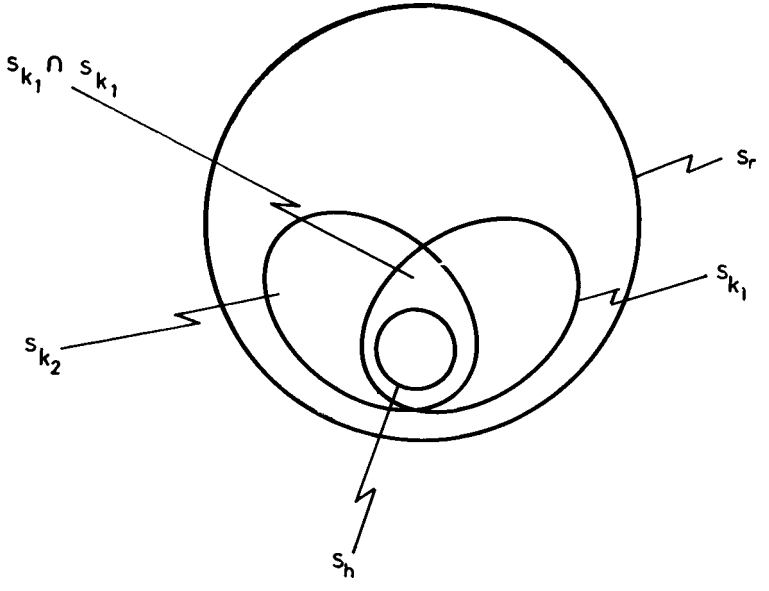


Fig. 10 Intersection of sets of keys

### 4.2 *Performing relational operations using the hashed bit map store*

*4.2.1 Queries involving the join of relations:* The method of using the hashed bit arrays store to perform joins must allow for the possibility of spurious keys appearing in the store. Consider the query to the join of *SP* and *PC* given earlier where the following question was asked: 'Which parts are supplied by Mullard to ICL?'

The previous method had the following steps:

(1) clear store
(2) store the parts supplied by Mullard using relation *SP*
(3) fetch the parts that are in the store supplied only to ICL using relation *PC*.

Line (2) stores the single part 'transistor'. If there are no spurious keys in the store then line (3) can exactly use this part and check that it is, in fact, supplied to ICL. Suppose, however, the hashed bit map is used: this might contain 'resistor' as a spurious key, although only 'transistor' was stored, using line (2). This spurious key is checked to see if it is supplied to ICL. Relation *PC* in Fig. 2 shows that the spurious key 'resistor' is indeed supplied to ICL and so line (3) will quite erroneously fetch 'resistor'.

The solution to the problem of spurious keys is to use a modified procedure, where sufficient information is retrieved during line (2) to prevent the errors in the output of line (3). The new procedure is given below with an additional line that removes spurious keys from the output by checking them against the true set of parts given by line (2) below.

(1) clear store
(2) store in the hashed bit map and in list $P'$ all the parts associated with Mullard in the *SP* relation
(3) place all parts into a list $P$ that have been stored in hashed bit form and are supplied to customer ICL
(4) eliminate from list $P$ those parts which are not in the list $P'$.

This example illustrates an important characteristic of the new joining technique. To eliminate the uncertainties generated by the spurious members in the hashed bit map, it is necessary, in line (2), to fetch back list $P'$ from the *SP* relation. Line (4) then uses list $P'$ to remove spurious parts from the list $P$.

Consider the problem of what maximum percentage of spurious keys can be allowed in the store. If keys are transmitted from one file to another then a reasonable level might be 10%. However, if coupling indexes between a succession of relations are stored sequentially, then the overall error should be less than 10%. Suppose the error level in the store is 3%, then 100 keys become 103 keys after they are recalled. If these 103 keys are stored again then they become $103 \times (1 + 3\%) = 106.09$ keys. This example shows that

small errors are approximately additive. Thus, assuming the usual number of relations to be joined is four (i.e. three links between them), then the store should have an error of not more than 3% for an overall error rate of about 10%. In considering detailed performance in the TODS paper a maximum error rate of 3% for the joining of relations was assumed for illustrative purposes.

*4.2.2 Queries involving the projection of a relation:* The proposed method of projection is the same as that used for the single bit map. It is, however, useful to understand the difficulties encountered when trying to use the hashed bit map for projection. A typical projection problem might be to remove the repetition in the set of keys (*a*, *a*, *b*, *b*, *c*, *c*). The projection technique remembers each value as it is retrieved and checks that the same value is not retrieved again. The hashed bit map technique is the same but can lose information as follows. Suppose the first element '*a*' in the list has been retrieved, and that although '*a*' is placed in the hashed bit map, '*b*' appears as a spurious key. When subsequent members of (*a*, *a*, *b*, *b*, *c*, *c*) are checked the system imagines that *b* is already in the hashed bit map, and so, in fact only (*a*, *c*) is actually retrieved. The only way the technique can be made effective is to make the chance of losing information in the hashed bit map extremely low. In other words, the chance of spurious keys being present must be very low.

An undetected error every 30 years might be considered acceptable for disc hardware. Bit maps can easily give a mean time between failures (i.e. lost keys) as great as 1 000 000 years. However, 3000 years is an acceptable figure and is assumed in the TODS paper. The database user would then have his normal mean time between failures decreased by approximately 1%. However, if a lower overall failure rate is needed, then a second check scan would be used. If the user is still unhappy, then he must come to regard projection as only approximate.

## 5 Discussion and conclusions

The aim of this paper was to show how relational operations could be performed by a content-addressable file store (CAFS). CAFS has a special store for intermediate results, which can either be addressed by coupling indexes or by field values. The methods of using these stores to perform relational operations have been considered, and various broad conclusions about their applicability can be drawn.

The single bit map involves increasing the size of a file by adding special coupling indexes. This slows down updates to a relation. However, queries involving projection or join can be answered quickly.

The hashed bit map requires no special indexes and so there is no file size increase. Update is therefore very straightforward. Sorting out ambiguities leads to an overhead on queries involving a join, and so answering queries is slower than when using the unhashed bit map. However, the hashed bit map enables fast execution of special selectors using non-Boolean functions.

The above characteristics suggest that the single bit map is most suitable for frequent queries on joins or projections. The ability of the hashed bit map to remember any key makes this ideal for the more unusual joins or projections, where maximum speed is not essential.

### References


1 BABB, E.: 'Implementing a relational database by means of specialised hardware', *ACM Trans. Database Syst.*, 1979, **4** (1), 1–29.
2 COULOURIS. G.F., EVANS, J.M. and MITCHELL, R.W.: 'Towards content addressing in data bases', *Computer J.*, 1972, **15** (2), 95–98.
3 MITCHELL, R.W.: 'Content addressable file store', Database Technology Conference, Online Conferences Ltd., April 1976.
4 PARKER, J.L.: 'A logic per track retrieval system', *Info. Proc. 71*, North Holland, 1971, 146–150.
5 OZAKARAHAN, E.A., SCHUSTER, S.A. and SMITH, K.C.: 'RAP – an associative processor for data base management', Proc. 1975 NCC, **44**, AFIPS Press, Montvale, New Jersey, USA, 379–387.
6 BUSH, J.A., LIPOVSKI, G.J., SU, S.Y.W., WATSON, J.K. and ACKERMAN, S.J.: 'Some implementations of segment sequential functions', Proc. 3rd Symp. Computer Architecture, Clearwater, Florida, USA, Jan. 1976, 178–175, available from ACM, New York.
7 LIPOVSKI, G.J. and SU, S.Y.W.: 'On non-numerical architecture', *Computer Architecture News (ACM)*, 1975, **4** (1), 14–29.
8 SU, S.Y.W., COPELAND, G.P. Jr. and LIPOVSKI, G.J.: 'Retrieval operations and data representations in a context-addressed disc system', Proc. ACM SIGPLAN-SIGIR Interface Meeting, *SIGPLAN Notices (ACM)*, 1975, **10** (1), 144–153.
9 SU, S.Y.W. and LIPOVSKI, G.J.: 'CASSM: a cellular system for very large data bases', Proc. Int. Conf. Very Large Databases, Framingham, Massachusetts, USA, Sept. 1975, 456–472.
10 LIN, C.S., SMITH, D.C.P. and SMITH, J.M.: 'The design of a rotating associative memory for relational database applications', *ACM Trans. Database Syst.*, 1976, **1** (1), 53–65.
11 BABB, E.: 'Hashed bit arrays store', UK Patent Application 27093/74, June 1974.
12 KNUTH, D.E.: '*The art of computer programming, vol. 3: sorting and searching*', Addison-Wesley, Reading, Massachusetts, 1973.


### Bibliography


1 BAUM, R.I. and HSIAO, D.K.: 'Database computers – a step towards data utilities', *IEEE Trans.*, 1976, **C-25**, 1254–1259.
2 CODD, E.F.: 'A relational model of data for large shared data banks', *Comm. ACM*, 1970, **13** (6), 377–387.
3 DATE, C.J.: '*An introduction to database systems*', 2nd edn., Addison-Wesley, Reading, Massachusetts, USA, 1977.
4 LEICHLICH, H.O., KARLOWSKY, I. and ZEIDLER, H.C.: 'Content addressing in databases by special peripheral hardware (suchrechner)', Workshop on Computer Architecture, Erlangen, DDR, May 1975.

# Patents relating to the ICL content-addressable file store CAFS

The following information was provided by ICL Patents, Stevenage. As the prefix GB indicates, all are British patents.

**Channel multiplexing** GB 14 97 676
M.W. Martin, R.W. Mitchell, D.R. Webb
Data read simultaneously from several tracks of a disc is time multiplexed into a single data channel connected in parallel to a number of time-shared comparison units.

**Multibit map** GB 14 91 706
E. Babb
The amount of storage necessary to hold search results is reduced by using two or more independently hash-addressed bit maps.

**Key channel** GB 14 97 677
R.W. Mitchell, D.R. Webb
Key channel has a 'substitute search' feature allowing two fields in same record to be compared.

**Remembered hits** GB 14 92 260
E. Babb
Data records are stored in a compressed form in which a data item occurring in one record is omitted from following records.

**Extended records** GB 14 97 678
E. Babb
Any one of a number of different markers can be selected to produce the 'end of record' signal, so that the effective boundaries of the records can be varied.

**Search evaluation unit** GB 14 97 679
R.W. Mitchell, D.R. Webb
A small array processor analyses the results of the search comparison units.

**Pilot format** GB 14 97 680
M.W. Martin
Data records are arranged sequentially from track to

track on a disc, rather than along each track, to reduce the amount of buffer storage in the data channel.

**Error feedback** GB 15 64 563

E. Babb

An improvement on the 'multibit map' case, in which a further store is used to reduce the number of spurious record selections which arise from the compressed storage of the search results.

# Notes on the authors

*E. Babb* CAFS file-correlation unit

Ed Babb obtained qualifications in Electrical Engineering and Computer Science from Imperial College. He then researched adaptive pattern-recognition systems at Cambridge University on secondment from Hawker Siddeley Dynamics. From about 1971, working in ICL, he researched speech recognition and information retrieval. His work on CAFS covered storage structures, architectures and query languages. He is now studying the application of mathematical logic to business systems and is currently the manager of the logic language project in the Systems Strategy Centre, Technical Directorate, Bracknell.

*L. Burnard* CAFS and text: the view from academia

Lou Burnard graduated from Balliol College, Oxford, in 1968 with first class honours in English and took an M.Phil. in 19th Century English studies three years later. He lectured in English at the University of Malawi for two years. He first encountered a computer when he joined Oxford University Computing Service in 1973, initially as an operator. Since 1976 he has worked in the area of database design and support, specialising in humanities applications, on which he has published several articles. He has been an active member of the BCS Database Administration Working Group (DBAWG) and of various ICL user groups, the most recent being the CAFS Working Party. Current projects include new CAFS-based text-searching software and implementation of a large museums database system.

*J.W.S. Carmichael* History of the ICL content-addressable file store (CAFS)

Hamish Carmichael has worked for 'ICL' since 1958, when he joined Powers-Samas after reading classics at Oxford. After experience of software and applications programming on a wide variety of early machines he helped in the formation of what is now Group Information Systems. For many years he was concerned with the design of ICL's in-house databases and with the applying of the company's hardware and software products to meet its own information requirements, and acquired a long-term interest in the principles

and practice of data management. Since 1980 he has concentrated on the development, exploitation and marketing of CAFS.

*C.E.H. Corbin* Creating an end-user CAFS service

Christopher Corbin BA(OU) has, after a four-year period as a Marine Radio Officer with Marconi Marine, a broad experience in the computer industry in both hardware and software, having worked on production, commissioning and support. Since 1975 he has worked for the Southern Water Authority where he has been the Technical Support Manager since 1977.

*G.McC. Haworth* The CAFS system today and tomorrow

Guy Haworth took a BA (Hons.) degree in Mathematics at Oxford and the Diploma in Computer Science at Cambridge. He then carried out research in the Cambridge Control Engineering Laboratory on the logical correctness and operational performance of computer systems. In 1972 he joined Ferranti's computer division, conducting research and development on computer architectures and real-time systems. He joined ICL in 1974 and, with a 1981–82 intermission at Cullinane (UK) has worked for the most part on application development tools in a consultant and marketing capacity. He is now Marketing Manager in ICL's Management Support Business Centre with special responsibilities for CAFS exploitation.

*A.T.F. Hutt* History of the CAFS relational software

After taking his first degree at University College, Durham, Dr. Hutt joined ICL in 1966 when he was seconded to Edinburgh University to work on the Edinburgh Multi-Access System (EMAS). In 1970 he started work on data-management software for the 2900 series and helped in the design of the VME/B catalogue, file store and record-access facilities. In 1973 he was seconded to Southampton University to work on the Relational Database Management System (RDMS) for which he was awarded a doctorate. Later, in ICL, he led a team which exploited Querymaster, the Personal Data System, CAFS-ISP and the Relational CAFS interface. Dr. Hutt is currently managing ICL's strategy for Man-Machine Interface.

*M.H. Kay* Textmaster — a document-retrieval system using CAFS-ISP

Michael Kay gained a Ph.D. from Cambridge for research on database management systems. After time with Sperry Univac he joined ICL in 1977 to work on IDMS, the Codasyl database system for the 2900 series. He led the IDMS design team in Bracknell between 1980 and 1982 and then moved to Reading to work on text retrieval and CAFS.

*N. Macphail* Development of the CAFS-ISP controller product for Series 29 and 39 systems

Neil Macphail is a Chartered Engineer and is a corporate member of both the Institute of Electronic & Radio Engineers and the British Institute of Management. He began working for ICL in 1964 and has since contributed to the development of KDF8, System 4, 7502 Terminal Systems, ME29 and 2900 and 3900 series, mainly in input/output development. Since 1980 he has managed the hardware development of CAFS with its associated microcode.

*R.M. Tagg* CAFS-ISP: issues for the application designer

Roger Tagg is an independent consultant working in the areas of database applications and end-user interfaces to data. After graduating from Cambridge in Mathematics he worked for the National Coal Board on operational research and subsequently in the Board's headquarters computer services (now Compower). He was for 12 years a consultant with Scicon, specialising in simulation, commercial data processing, databases and lecturing/training. He has been independent since 1980 and has worked for a variety of industrial and government clients, including ICL users. He was formerly a representative on the Codasyl Data Description Language Committee and also on its UK (British Computer Society) associate group, the Database Administration Working Group (DBAWG). He was the founder and is the chairman of the BCS End-user Systems Group (formerly the Query Language Group).

*D. Walker* Secrets of the sky: the IRAS data at Queen Mary College

David Walker studied mathematics as an undergraduate at Jesus College, Cambridge. After graduating in 1976 he was awarded an M.Sc. in Astrophysics at Queen Mary College, University of London, where he later went on to gain a Ph.D. in physics. In 1982 he began working on the Infra-Red Astronomical Satellite (IRAS) project and has spent several months at NASA's Jet Propulsion Laboratory in California, designing data reduction and analysis systems and assessing the performance of the satellite. He is now a postdoctoral research assistant in the Theoretical Astronomical Unit at QMC, where he is engaged in the scientific interpretation of the IRAS data.

*P.R. Wiles* Using secondary indexes for large CAFS databases

Peter Wiles has a B.Sc. and M.Sc. in Computer Science from Manchester University. He joined ICL in 1971 and worked at Kidsgrove on the development of VME, eventually joining Ostech as a strategist. He moved to Marketing in 1977 and joined the EEC Project in Luxembourg, where he was

responsible for the production of software tools to assist the conversion of the Commission's workload from IBM 370 to ICL 2900 systems. He moved to the Inland Revenue project in 1981. He is now the Product Exploitation Manager for this project, and among his responsibilities is encouragement of the use of CAFS by the Inland Revenue.

ICL TECHNICAL JOURNAL

Pages contained in each issue

(1) 1–116
(2) 117–220
(3) 221–348
(4) 349–520

# Subject index
## Volume 4

# Author index
## Volume 4