Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

359

D. Gawlick M. Haynie A. Reuter (Eds.)

High Performance Transaction Systems

2nd International Workshop
Asilomar Conference Center, Pacific Grove, CA, USA
September 28–30, 1987
Proceedings

Springer-Verlag
Berlin Heidelberg New York London Paris Tokyo

Editors

Dieter Gawlick
Digital Equipment Corporation
10500 Ridgeview Ct., Cupertino, CA 95014, USA

Mark Haynie
Amdahl Corporation, 1250 East Arques Ave.
P.O. Box 3470, Sunnyvale, CA 94088, USA

Andreas Reuter
Institut für Parallele und Verteilte Höchstleistungsrechner
Universität Stuttgart
Azenbergstr. 12, Herdweg 51, D-7000 Stuttgart 1, FRG

Program Committee

C. Mohan, IBM Almaden Research Center (Chairman)
Dieter Gawlick, Digital Equipment Corporation (Workshop Chairman)
Phil Bernstein, Digital Equipment Corporation
Jim Gray, Tandem Computers
Pat O'Neil, Computer Corporation of America
Andreas Reuter, University of Stuttgart
Tom Scrutchin, Boole & Babbage

CR Subject Classification (1987): C.1.2, C.2, C.4, D.2.7, D.4, E.0, H.2, K.6

ISBN 3-540-51085-0 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-51085-0 Springer-Verlag New York Berlin Heidelberg

Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
2145/3140-543210 – Printed on acid-free paper

Contents

Foreword

In Fall 1985, the first event called 'International Workshop on High Performance Transaction Systems' took place at Asilomar. It was an informal gathering of users, developers and researchers from the field of large transaction processing systems, and the idea was to discuss current topics, exchange experiences, explore future developments, in other words: define what was needed to achieve high performance with transaction oriented systems.

Many discussions at this first workshop were triggered by the problem of multi–processor database architectures, i.e. the question of how to couple processors in order to efficiently run transactions against one (logically) centralized database. The distinction of architectures into 'shared everything', 'shared disk', and 'shared nothing' was invented for a position paper at that workshop by Mike Stronebraker.

Since many participants felt the first one was a success, it was decided to repeat the workshop two years later. In 1987, there were more technical papers in the formal sense, although discussions and ad hoc sessions still played a major role. It was quite interesting to see that a number of suggestions that had been made two years earlier had meanwhile influenced some products one way or the other; some issues that had been discussed were settled meanwhile (much of the sharing controversy, e.g.), and new topics had come up, some of which have rarely been discussed in the context of transaction processing systems elsewhere.

It looks like many of the problems identified during the '87 workshop will determine the future development of transaction systems and distributed high performance systems in general for many years to come. So the organizers of HPTS '87 felt encouraged to collect the papers presented at the workshop in order to make them accessible to a wider audience of interested developers and researchers. Since some of the contributions represented work in progress, the authors agreed to prepare revised and updated versions of their papers for this publication. This accounts for the long delay between the event itself and the publication, but on the other hand it provides the reader with a state–of–the–art account of transaction processing topics.

The book is organized according to the major sections of the workshop. In the networking section, we find an analysis of two of the major 'paradigms' in networking, ISO/OSI and SNA, from the perspective of transaction processing. First,

M. Bever, M. Feldhoffer and S. Pappe describe the ongoing efforts of the ISO to incorporate transaction protocols into their overall open systems architecture. In particular, they describe the ideas discussed in the CCR- (commitment, concurrency and recovery) group with respect to transaction models, error handling, etc. They also present examples of applications based on CCR-protocols, especially an RDA (remote database access) interface.

In the second paper, Wayne Duquaine presents an analysis of LU 6.2, which is SNA‘s transaction processing protocol. He puts some emphasis on the use this interface can make of intelligent terminals like PCs to handle major portions of the protocol. Certainly this will be a very important aspect in the design of the next generation of TP systems.

It should make an interesting exercise for the reader to compare the basic assumptions of CCR on one hand and LU 6.2 on the other. The result will be that they do not look all too different. So the next question should be whether these protocols contain all we need for future transaction processing, or whether they just codify the way it is done now. This was discussed heavily at the workshop, but - of course - there is no answer yet.

Both papers agree on the necessity to include network management into the discussion about networking standards.

The next section is comprised of a description of four different transaction processing and database systems.

It starts out with Pat O‘Neil‘s summary of Model 204, a database management system marketed by Computer Corporation of America. The paper makes it very clear that this system is optimized for handling complex selection predicates, which is not typical for today‘s mainstream online transaction processing (OLTP) applications, but might well be in the future.

The complete opposite - from that perspective - is Tandem‘s NonStop SQL, which is described in the next contribution. It was developed with OLTP explicitly in mind. In the paper, all important design and implementation issues are covered, and the last part describes the results of a benchmark based on the Debit/Credit database and workload definition. It is worth noting that one of the first commercially available distributed relational systems is at the same time one of the top performers in terms of OLTP workloads.

A significantly different approach to handling the same type of workload (online banking transactions) is covered in Frank Bamberger‘s paper on Citicorp‘s transaction processing system. This company just completed the transition from one unique installation, consisting of a series of self-developped machines, to another unique configuration based on TPF with a couple of own protocols and application software on top. The paper might be of particular interest for those interested in the problems of either setting up or migrating large transaction oriented applications.

The last paper in the systems section is authored by Steve Hobson and describes the implementation of ALCS, which basically is a version of TPF running under MVS/XA. Considering that TPF (and its precursor, ACP) is an 'old' system by the standards of our discipline it may seem surprising that substantial development is still being invested into extending and adapting it. However, there is a demand for it, as is shown by at least one large project for building a new airline reservation system, which uses ALCS. And this also makes sense from a technical viewpoint, as you can see from the implementational descriptions. Such systems offer the basic mechanisms for very large OLTP systems, which more 'general purpose' operating systems etc. don't, like very fast message routing and context switching, simple but efficient fault tolerance mechanisms, etc.

The research section contains four very different contributions which are fairly representative of the type of problems in transaction systems investigated in the research community.

Gary Herrman and Gita Gopal present an architecture for achieving very high transaction rates, which they call Datacycle. The idea is highly unconventional, assuming that the data is permanently broadcast over a very high-bandwidth medium to all processing sites. Each site can 'suggest' updates, which are decided upon by some authority. If they are accepted (i.e. they are serializable), the updates will be visible in the next broadcast round.

The authors claim - and support it by some strong arguments - that conventional architectures are not likely to deliver the throughput required in some very high performance transaction systems, so radically new architectures might well deserve much more attention than they get these days - at least in the area of TP systems.

The next paper by Kenneth Salem, Hector Garcia-Molina, and Rafael Alonso deals with a very old, very hard problem of transaction processing, which is the co-existence of short transactions and very long transactions, be it of the batch-type, or for modelling long lived activities. The protocol described in this paper is based on the idea of releasing locks before the end of a 'long' transaction on those objects that will not be accessed again. Other transactions can then access these objects, must, however, observe additional protocols in order to avoid inconsistencies.

Gerhard Weikum's paper discusses the implications of a nested transaction model when applied to the implementation of a transaction system, with special emphasis on multi-layered concurrency control. Although such techniques are already used in existing systems (e.g. tuple locking), the formal framework presented here might prove very helpful in understanding the structures of such protocols and in designing more reliable (and efficient) implementations of layered synchronization techniques.

The last paper by Alfred Spector and his students gives an account of design, implementation and performance characteristics of Camelot. This is a distributed transaction management system based on Mach, a Unix-compatible operating system that has been extended by a number of features which are indispensable for implementing high performance transaction systems. Some of the key features are shared memory and cheap processes (tasks). The performance figures illustrate quite clearly that, given the right basic mechanisms, competitive performance can be achieved (using Debit/Credit-like measures) in an environment of Unix workstations.

Since this was a high performance system workshop, of course there has to be a performance section.

It starts with a performance evaluation of Teradata's DBC1012 by Dave DeWitt, Marc Smith and Haran Boral. The paper describes a fairly elaborate measurement with a number of parameters, functions investigated, and accordingly contains a whole range of results which cannot easily be summed up. One of the key observations is that the scheme used by this machine for achieving parallelism, i.e. the hash-based distribution and storage allocation of tuples seems to cause performance problems under special circumstances. However, readers interested in that matter should give the paper a careful reading.

The paper by Anupam Bhide and Mike Stonebraker is an attempt to add some quantitative arguments to the 'shared whatever' discussion. It describes a simulation-based comparison of a shared everything and a shared nothing architecture, including CPU consumption, message costs, disk accesses, etc. Intra-transaction parallelism is also considered. The general result is the superiority of shared everything, especially if there are response time constraints and/or hot spots. Load balancing is equally important for both architectures, and transaction internal parallelism improves the performance in either case.

The last paper by Pat Helland and others describes a special technique for increasing throughput in Tandem‘s transaction manager, namely group commit. The idea is to collect commit records in the log buffer (rather than forcing them individually) and to write a batch of them in just one I/O. Obviously, this works well under high transaction rates, but under conditions of low load, the buffer must be forced explicitly in order to keep response time within limits. The paper explains an analytic model for determining the optimal bundling factor and timer setting, which has been used for incorporating a simple balancing strategy into TMF.

As one can see from the agenda, there were more presentations than the ones mentioned so far. However, they did not go with a full paper; these were only foils or collections of one-liners, which is why they are not included in this book. But since many of them raised quite interesting issues or succeeded in stimulating wild discussions, I will try to summarize them based on the scarce material I am left with in some cases. So if something of what follows sounds weird, you will most likely have to blame my summary rather than the original contender.

In the first round, Harald Sammer and Dieter Gawlick discussed the disadvantages of 'shared nothing‘ and 'shared everything‘ systems, respectively. This was particularly interesting, because both have profound architectural experience with the type of systems they had to criticize.

Harald Sammer gave a list of disadvantages inherent to shared nothing systems (i.e. distributed systems in a very general sense) as follows:
Such systems suffer from message overhead, since messages are the only means for processors to cooperate. Without special precautions, which are not supported in all current communication architectures, the number of sessions between processes grows as a square function of the number of nodes, and very quickly gets out of hand. Handling messages through layered communication protocol increases pathlength.
Synchronization based on messages only is much harder than in a shared memory environment. This applies to logs as well as to concurrency control.
Shared nothing systems require efficient load balancing (which is hard) and – as a complementary problem – application decomposition. Last not least, debugging such systems is extremely difficult.

However, on a large scale each system will eventually be a 'shared nothing' system, so all of these problems have to be managed anyway.

Dieter Gawlick's characterization of 'shared everything' architectures started at this very point by saying that it is an electronic complex connected to one storage hierarchy, in which all the required data reside, such that each unit of work can be executed completely in one electronic complex. What follows immediately from this definition is that only dumb terminals can be part of such a system and that no online interaction between two 'shared everything' systems can be part of any unit of work. Whereas these systems appear to be superior for simple, isolated applications, they make both implementation and changing of complex applications hard (due to their monolithic attitude) and tend to become unmanageable as they grow. The same is true for performance, in that in complex, integrated applications shared everything means one has to use high cost, high performance components, because there is no way of functional distribution. High level of multi-tasking implies problems in maintaining constant response times for routine tasks like text processing, etc. Shared everything systems have single points of failure, which causes serious availability problems; disaster recovery is nearly impossible. Shared nothing systems, on the other hand, still cannot keep data as corporate assets.

Whereas these position statements were made from the perspective of classic OLTP systems, Sam DeFazio and Charles Greenwald presented a highly 'exotic' application, the LEXIS/NEXIS Information Retrieval System, run by Mead Data Central. Their transactions do full text search over (at that time) 200 billion characters of raw text with fairly complex search criteria. They have to support a peak load of 8.000 transactions/hour, with a response time target of 12 secs. Although the database is fully inverted (except for 'noise' words) any such transaction does 400+ I/Os and has a pathlength of more than 15 million instructions. Given this, the workload requirements are really tough. The presentation made it very clear, in which points current mainframe architectures and OS structures are inadequate for such applications – and the key problem areas are: No functional specialization, and too little support for substantial parallelism within one application.

Next were two presentations on network management systems, NetMaster (Cincom) and NetView (IBM). This was particularly interesting to the audience, since – as I mentioned before – network management was generally felt to be an area needing much more attention and work. Of course, it is impossible to summarize these product descriptions. The general impression, though, can be put as follows: Both systems - to a certain degree - implement their own type of network on top of whatever the underlying network is, provide their own TP-monitor etc. and effectively manage that. What would be required, on the other hand, is an integration of communication protocols and network management services.

In addition to the long papers on OSI and LU 6.2, there was a position statement by Andreas Reuter weighing connection oriented vs. connectionless communication protocols from the perspective of transaction processing. He basically argued that the increase in quality of communication due to sessions or similar concepts are not really helpful for implementing transaction protocols. They are 1 : 1, whereas transactions can involve many nodes, which means a two-phase commit has to include several sessions. Put the other way: The implementation of transaction protocols does not become much simpler if you have sessions rather than datagrams. One counter argument was that with datagrams one has to go through authentication for each packet.

In the systems section, there was a contribution describing IBM's ONEKAY benchmark with IMS Fast Path, which unfortunately is not available as a full paper. The workload reflects a credit card application i.e. card authorization, credit limit check, debit processing and reporting of lost or stolen cards. There were four databases, 1 DEDB holding account information (13 M records), 1 MSDB holding exception cards (14 MB), 1 MSDB for establishment activity counters (2K records), 1 DEDB for added exception cards (500 records).
The transaction type bearing most resemblance to the Debit/Credit benchmark transaction was DEBIT, which did the following:
Get MSG from terminal; fetch and update account database root; put the transaction in the journal (dependent segment); update establishment database; reply to the terminal.
The benchmark was run on a 3090-400 with 128 MB of memory (+ 256 MB ESD), 96 channel paths and 86 volumes for the database (the DEDB-part).
The DEBIT transaction peaked at 933 TPS, with 95 % CPU utilization and .34 secs average transaction transit time.
Some interesting performance bottlenecks in IMS and VTAM were discovered during the benchmark (and removed, of course).

Don Haderle gave a brief overview of the DB2 development, with an emphasis not so much on technical details, but on the general design decision and the parameters having influenced them. It was an exceptionally interesting account of what shapes a new product and how it is tied to the environment, but based on just the presentation it is impossible to give a fair summary of what was said.

Probably the most vivid session was Ed Lassetre's attempt to outline the 'Future DB Mechanisms for Transaction Processing'. On his first slide, he tried to classify applications into three categories: Short transactions with few data and simple processing requirements, i.e. current OLTP style; second were transactions operating on complex data structures (objects) with many links and references, like in CAD, and with moderate processing requirements; the third group was characterized by very complex computations on vast amounts of data, like in deductive systems.
He then asked whether we would have different types of systems for each class, with the usual problems for 'mixed' applications, or if super-systems doing equally well in all cases would be conceivable, or if there would be generic architectures that can be configured for each application profile. As a matter of fact, Ed Lassetre did not get beyond this first slide. Discussion heated up very quickly and covered all the ground there was. It is impossible to repeat all the arguments that came up; but quite obviously there were representatives from different camps, i.e. different view points on what transaction processing means and what a transaction processing system should deliver. For example, some participants contended that application types 2 and 3 are not transaction-related, and transaction systems had nothing to do with that. Others claimed that at least category 2 needed support by transaction-like control sturctures, but again there was the argument on whether this was just an issue of data models and user interfaces, or if this had further impact on the underlying system and its implementation. On the other hand, it became quite clear that most of the participants, users, developers etc. were confronted with the problem of either extending transaction services to other applications or of integrating them one way or another. So this question will be around for a couple of years.

In retrospect, this workshop will probably mark the moment where transaction processing systems made the 1KTPS-technology available for production environments. It also indicated increasing interest in making transactions a much more fundamental paradigm of processing than they are now. The big challenge in the future will be to shape systems which are strictly transaction oriented down to the very primitives of the OS, but still support all conceivable applications (plus the new ones) in a better way than today's systems can.

Andreas Reuter