The Asilomar Report on Database Research

by
Phil Bernstein, Michael Brodie, Stefano Ceri, David DeWitt, Mike Franklin,
Hector Garcia-Molina, Jim Gray, Jerry Held, Joe Hellerstein, H. V. Jagadish,
Michael Lesk, Dave Maier, Jeff Naughton, Hamid Pirahesh, Mike Stonebraker, and Jeff Ullman

Executive Summary

The database research community is rightly proud of success in basic research, and its remarkable record of technology transfer. Now the field needs to radically broaden its research focus to attack the issues of capturing, storing, analyzing, and presenting the vast array of online data. The database research community should embrace a broader research agenda -- broadening the definition of database management to embrace all the content of the Web and other online data stores, and rethinking our fundamental assumptions in light of technology shifts. To accelerate this transition, we recommend changing the way research results are evaluated and presented. In particular, we advocate encouraging more speculative and long-range work, moving conferences to a poster format, and publishing all research literature on the Web.

1. Introduction

On August 19-21, 1998, a group of 16 database system researchers from academe, industry, and government met at Asilomar, California to assess the database system research agenda for the next decade. This meeting was modeled after similar meetings held in the past decade[1]. The goal was to discuss the current database system research agenda and, if appropriate, to report our recommendations. This document summarizes the results of that meeting.

The database system research community made major conceptual breakthroughs a decade ago in the areas of query optimization, object-relational database systems, active databases, data replication, and database parallelism. These ideas have been transitioned successfully to industry, and the research community should be proud of its recent successes.

There is reason for concern, however, since the community is largely continuing to refine these ideas, in what has been characterized as "delta-X" research. True, there is a kind of incremental research in which a series of steps build upon previous steps, leading to long-term, important innovations; it is not this sort of activity that concerns us. However, "delta-X" research often has a short-term focus, namely improving some widely understood idea X. Often, the underlying idea X already appears in some product, hence this sort of "delta-X" research can be done by industrial development labs and startups backed by venture capital.

We encourage the database research community to eschew the latter kind of "delta-X" research. Let's broaden our focus to explore problems whose main applications are a decade off, leaving short-term work to other organizations. Funding agencies and program committees should encourage this kind of forward-looking research by explicitly recognizing that highly innovative, although speculative, work should generally be ranked above more polished work of an incremental, short-term nature.

The fundamental database system issues have changed dramatically in the last decade. As such, there are ample new issues for database system research to investigate. Therefore, we call for a redirection of the research community away from incremental work and toward new areas.

The remainder of this report is organized as follows. Section 2 discusses the driving forces that fundamentally change the database system research agenda. This discussion motivates the specific issues, which we propose as a database system research agenda in Section 3.

To help focus the database system research agenda on long-range problems, we present a "grand challenge" research problem with a ten-year goal in Section 4.

Section 5 proposes radical changes to the way database system conferences and journals judge and

---

[1] Laguna Beach meeting of 1988 [SIGMOD Record 18(1): 17-26], Lagunita meetings of 1990 and 1995 [SIGMOD Record 19(4):6-22, SIGMOD Record 25(1):52-63] http://www.acm.org/sigmod/record/issues/9603/lagunita.ps. ACM 1996 meeting "Strategic Directions in Database Systems---Breaking Out of the Box," ACM Computing Surveys 28(4): 764-778. http://www.acm.org/surveys/sdcr/

present research results. The current process and organization encourages incremental results and discourages pioneering work -- this process must change if we want to encourage radically new ideas.

2. Driving Forces

Three major forces are shaping the proposed focus of database system research:

1. The Web and the Internet make it easy and attractive to put all information into cyberspace, and makes it accessible to almost everyone.
2. Ever more complex application environments have increased the need to integrate programs and data.
3. Hardware advances invalidate the assumptions and design decisions in current DBMS technology.

The reader is certainly aware of these trends, but we recapitulate them here to motivate our assertion that the database research agenda needs to be redefined in terms of these new assumptions.

2.1. The Web Changes Everything

The Web and its associated tools have dramatically cut content creation cost, but the real revolution is that the Web has made publishing almost free. Never before has almost everyone been able to inexpensively publish large amounts of content. The Web is the major platform for delivery of applications and information. Increasing amounts of available bandwidth will only accelerate this process.

This is good news for database systems research: the Web is one huge database. However, the database research community has contributed little to the Web thus far. Rather than being an integral part of the fabric of the Web, database systems appear in peripheral roles. First, database systems are often used as high-end Web servers, as webmasters with a million pages of content invariably switch to a web site managed by database technology rather than using file system technology. Second, database systems are used as E-commerce servers, in which they are used in traditional ways to track customer profiles, transactions, billing, and inventory. Third, major content publishers are using or evaluating database systems for storing their content repositories. However, the largest of the web sites, especially those run by portal and search engine companies, have not adopted database technology. Also, smaller web sites typically use file system technology for content deployment, using static HTML pages.

In the future, we see the web evolving to managing dynamic content, not static HTML pages. For example, catalog retailers do not simply transform paper catalogs into a collection of static HTML pages. Instead, they present an electronic catalog that allows consumers to ask for what they want without browsing: for example, does the vendor sell all-cotton teal polo shirts in size large. Retailers want to provide personalized mannequins that show how the clothing might look on you. Personalization requires very sophisticated data models and applications. Supporting this next generation of web applications will require very sophisticated database services.

Furthermore, HTML is being extended to XML, a language that better describes structured data. Unfortunately, XML is likely to generate chaos for database systems. XML's evolving query language is reminiscent of the procedural query processing languages prevalent 25 years ago. XML is also driving the development of client-side data caches that will support updates, which is leading the XML designers into a morass of distributed transaction issues. Unfortunately, most of the work on XML is happening without much influence from the database system community.

Web content producers need tools to rapidly and inexpensively build huge data stores with sophisticated applications. This in turn creates huge demand for database technology that automates the creation, management, searching, and security of web content. Web consumers need tools that can discover and analyze information on the Web.

These trends are opportunities for database researchers to apply their skills to new problems.

2.2. Unifying Program Logic and Database Systems

Early database systems worried only about storing user data, and left program logic to other subsystems. Relational database systems added stored procedures and triggers as an afterthought -- for performance and convenience. Current database products let applications store and activate database system procedures written in a proprietary programming language. The emergence of object-relational techniques, combined with the increasing momentum behind Java as a standard language, allow database systems to incorporate program logic, written in a standard programming language and type system. As such, database systems are on a transition path from storing and manipulating only data to storing and manipulating both logic and data.

However, there is still much work to be done. Repositories are typically databases of program logic. The requirements of repositories, such as version control and browsing are not well-served in most current systems. Obviously, code is not a first class object and co-equal to data in current database systems.

Continuing this transition is of crucial importance. Large enterprises have hundreds, sometimes thousands, of large-scale, complex packaged and custom applications. Interoperation between these applications is essential for the flexibility needed by enterprises to introduce new web-based applications services, meet regulatory requirements, reduce time to market, reduce costs, and execute business mergers. Advances in database technology will be required to solve this application integration problem.

Today, system integration of large-scale applications is largely addressed by software engineering approaches, with much attention to development process, tools, and languages. The database field should have more to contribute to this area. This requires that database systems become more application-aware. Object-relational techniques are part of the answer, but so are better techniques for managing descriptions of application interfaces, and higher-level model-driven tools that leverage these descriptions to help integrate, evolve, migrate, and replace application systems — both individual systems and groups of systems that function as a single system.

2.3. Hardware Advances: Scale up to MegaServers and Scale Down to Appliances

Moore's law will operate for another decade: CPUs will get faster, disks will get bigger, and there will be breakthroughs in long-dormant communication speeds. Within ten years, it will be common to have a terabyte of main memory serving as a buffer pool for a hundred-terabyte database. All but the largest database tables will be resident in main memory. These technology changes invalidate the fundamental assumptions of current database system architectures. Data structures, algorithms, and utilities all need re-evaluation in the context of these new computer architectures.

Perhaps more importantly, the relative cost of computing and human attention has changed: human attention is the precious resource. This new economics requires that computer systems be autoeverything: autoinstalling, automanaging, autohealing, and autoprogramming. Computers can augment human intelligence by analyzing and summarizing data, by organizing it, by intelligently answering direct questions and by informing people when interesting things happen.

The explosion in enterprise-wide packaged applications such as SAP™, Baan™, and Peoplesoft™ puts terrific pressure on database systems. It is quite common for users to want database system applications with 50,000 concurrent users. The computing engines and database system on which such applications are deployed must provide orders of magnitude better scalability and availability.

If technology trends continue, large organizations will have petabytes of storage managed by thousands of processors -- a hundred times more processors than today. The database community is rightly proud of its success in using parallel processing for both transaction processing and data analysis. However, current techniques are not likely to scale up by two more orders of magnitude.

In ten years, billions of people will be using the Web, but a trillion "gizmos" will also be connected to the Web. Within the next decade there will be increasingly powerful computers in smart-cards, telephones, and other information appliances. There will be substantial computing engines in the portable organizers (e.g., Palm Pilots™) and cell phones that we carry. Moreover, our set top boxes and other home appliances will be substantial computers. Smart buildings will put computers in light switches, vending machines, and many appliances. Each piece of merchandise may be tagged with an identity chip. All these information appliances have internal data that "docks" with other data stores. Each gizmo is a candidate for database system technology, because most will store and manage some information.

Because of gizmos, we foresee an explosion in the size and scale of data clients and servers -- trillions of gizmos will need billions of servers. The number, mobility, and intermittent connectivity of gizmos render current client-server and three-tier software architectures unsuitable for supporting such devices. Most gizmos will not have a user interface and cannot have a database administrator -- they must be self-managing, very secure, and very reliable. Ubiquitous gizmos are a major driver for the research agenda discussed in the next section.

3. A Proposed Research Agenda

This section discusses research topics that merit significant attention. The driving forces discussed above motivate each of these research topics. For simplicity, we group the topics under five main themes, and discuss each in turn.

3.1. Plug and Play Database Management Systems

We use the phrase *Plug and Play* in two ways. First, since gizmo databases will not have database administrators, a gizmo database must be self-tuning. There can be no human-settable parameters, and the database system must be able to adapt as conditions change. We call this **no knobs operation**. The database research community should investigate how to make database systems knob-free. The cornerstone of this work is to make database systems self-tuning, i.e. to remove the myriad of performance parameters that are user-specifiable in current products. A further portion of this work is to deal with physical database design, for example the automatic index selection techniques that have received some attention in recent research and products. More generally, the system should also help with logical database design (e.g. tables and constraints), and with application design, automatically presenting useful reports and utilities. To guarantee good behavior over time, a no-knobs system must adapt as conditions change.

Although we do not wish to specify a particular solution, an encouraging approach is to have the database system remember all traffic that it processes. Then, a *wizard* embedded in the database system with detailed tuning knowledge examines this traffic and autotunes the system. A side-benefit is that traditional commercial database systems become vastly easier to administer. Since most organizations do not have enough database administration talent to go around, no-knobs operation would help them enormously.

A second aspect of *Plug and Play* database systems deals with information discovery. As noted earlier, the Web is a huge database. Moreover, most commercial enterprises are having trouble integrating the "islands of information" present in their various systems. It should be possible to attach a database system to a company network or the Internet, and have the database system automatically discover and interact with the other database systems accessible on the network. This is the data equivalent of operating system support for hardware, which discovers and recognizes all accessible devices.

This information discovery process will require that database systems provide substantially more metadata that describes the meaning of the objects they manage. In addition, the database system must have a rich collection of functions to cast data from one type to another. It is reasonable to expect that there are other approaches to information discovery as well.

3.2. Federate Millions of Database Systems

Billions of web clients will be accessing millions of databases. Enterprises will set up large-scale federated database systems, since they are currently investing enormous resources into many disparate systems. Moreover, the Web is one large federated system. We must make it easy to integrate the information in these databases. There are several major challenges in building scalable federated systems.

First, we need query optimizers that can effectively deal with federated database systems of 1000 or more sites. It is an absolute requirement that each site in such a system be locally autonomous. Therefore, a federated query optimizer cannot simply construct an optimal plan, because various sites must be empowered to refuse to perform their piece. Local constraints may make the globally optimal plan infeasible. In addition, the load on the various sites may change. A traditional static cost-based optimizer computes an optimal plan assuming that the query is the only task running on the network. This plan is not "load aware", and even if it were, the load might change between compile and run time, or during run time. In a dynamic network, optimizers must adapt to changing loads. In a federated database system there may be replicas at various sites, and the quality (timeliness) of the replicas may vary. An optimizer must be able to deal with such quality-of-service issues. For all of these reasons, it is time to rethink the traditional static-cost-based approach to query optimizers in this new environment.

A second aspect of federated database systems is one of the semantics and execution of queries. A user might issue a query to a 1000-site federated database such as:

```
"Find the average enterprise-
 wide employee salary."
```

Traditional database systems are programmed to give the exact answer to this inquiry, perhaps after computing for a long time. A better model has the database system view this as an evidence accumulation process. The database system should develop a coarse answer quickly and then refine it over time, stopping when the user decides that the answer is "good enough." Of course, this requires substantial changes to a query optimizer and execution engine, but it also requires a synthesis of statistical estimation techniques with data delivery and user interfaces.

Imprecise information will not only appear as the output of queries; it already appears in data sources as well. The evidence accumulation paradigm has even wider utility in this regard. Consider a user submitting a query such as:

```
"Are there any really good Italian restaurants within 5 miles of where I live?"
```

There may be 10 or more restaurant review databases that have information on Italian restaurants, along with perhaps several geographic databases. Hence, this query presents an interesting database integration problem. There is no exact answer to this query, since each critic is entitled to his own opinion. The query engine must treat this as an evidence accumulation problem, albeit an even less clearly specified one than the previous example. Progressive refinement should be applicable in this universe as well.

A third aspect to federation is tools that assist the integration process itself. It should be easy for a system administrator to add his system to a larger federated system. If a web-based clothing retailer decides to offer their travel clothing to an online travel agent, then the clothing order and billing systems must federate with the travel agent systems. Automating this integration activity requires a database of application and database interface definitions merged into a coherent whole, with tools that help the engineer reconcile the new system being put in place. OMG's Unified Modeling Language standard is a step toward expressing these definitions, but more semantics must be captured in a computable form for tools to support improved automation.

3.3. Rethink Traditional Database System Architecture

The technology trends of Section 2 allow users to implement larger and larger database system applications. This has led to a multitude of shared memory, shared disk (cluster), nonuniform memory (NUMA) and shared-nothing cluster architectures. Current database systems have been especially successful with shared nothing systems since these have better scalability characteristics. Computer clusters also leverage commodity components, and so can be much less expensive. In a large cluster, the database system optimizer must deal with multiuser load balance, availability of disk space, and constraints on feasible plans and replicas. Most of the reasons for rethinking optimization in federated database systems (Section 3.2) also arise in this context.

In addition, the typical computing engine may have one terabyte of main memory. "Hot" tables and most indexes will be main-memory resident. This will require storage architectures to be rethought. For example, B-trees are not the optimal indexing structure for main memory data. Also, the current buffering, recovery, and concurrency strategies of commercial database systems may be inappropriate.

Furthermore, while disk capacities are improving very quickly, seek times are improving relatively slowly. Hence, the amount of data that can be transferred to main memory during an average seek time is rising very quickly. Put differently, the cost of a seek relative to the transfer of a byte of data is rising quickly. This requires storage architectures that are much more serious about disk arm optimization. Also, "arm wasting" architectures, such as RAID 5, may be inappropriate in the future.

Most organizations need continuous system operation. Designing a software system that never fails requires remote replicas and dynamic reconfiguration. It is not clear whether remote replicas should be handled at the disk level using RAID ideas, or at the database system level by moving the database system log and rolling it forward at the remote site.

New applications, including satellite imagery and digital television archives, require very large databases that are measured in petabytes or exabytes. Such applications may be enabled when disk storage becomes cheap enough to deal with the volume of required data in a standard 2-level memory hierarchy. Alternately, it is possible that a new tertiary storage device perhaps based on holographic techniques will become available. So, three level memory hierarchies are a definite possibility. Providing exabyte storage in multitier architectures, including replication and backup, is a considerable challenge.

Last, the popularity of three-tier (thick middleware) application architectures is increasing. In this world, there is only one program (the database system) running at the server level and only one program (the application server) running in the middle tier. Both must support thousands of connections.

Optimizing database systems (and operating systems) for this environment is a challenge.

In summary, the fundamental architecture of database systems has been around for nearly 20 years. We believe it is time to rethink most of the basic architectural assumptions in light of the environment that will be available in the year 2010.

3.4. Smart-Data Unify Process and Data in Database Systems

There are several ideas that should be investigated under the rubric of making application logic a first class citizen in future database systems. First, a possible model for the description of the application would be as a workflow of business rules. Such workflow systems are now available from many vendors as application-level frameworks. It seems possible to compile workflow diagrams into a collection of database triggers and alerters that would run inside an active database system. Running data-intensive workflows inside the database system is substantially faster than running them outside the system. However, workflow support requires a system capable of scaling to thousands of triggers. Current implementations scale only to a few triggers. Substantial research and experimentation is needed to scale up trigger systems by three orders of magnitude.

A scalable trigger system has additional benefits, because there are applications that need a large number of conventional triggers on data elements. For example, users of stock market systems wish to be notified when a particular condition becomes true, such as a particular stock reaching some price threshold. A scalable trigger system could support such applications where millions of triggers are defined on the data. Of course, such a trigger implementation must work on a shared-nothing or even federated system. Efficiently supporting large collections of distributed triggers is an open question.

A second issue concerns logic in the conventional sense of procedures in a given programming language. Components are a popular way of expressing such logic, and such components should be supported inside the database system. Unfortunately, there is not yet a component Esperanto. CORBA, OLE, Enterprise Java Beans (EJB), and Jini may all become popular. Supporting several (incompatible) component models inside an Object-Relational database system appears to be a daunting challenge. Nonetheless, it is incumbent on the database community to help evolve these models to support types and procedures well-integrated with the database system.

A third issue concerns visual programming methodologies. Many data designers use a diagramming framework to specify data and application design. These powerful tools can both model and automatically generate the application. If components are inside the database system, then such methodologies must be evolved to deal with Object-Relational database systems. It is an interesting challenge to consider a visual design tool that addresses all these system design aspects together.

A last issue in this arena concerns persistent programming languages. There are many applications where SQL is the dominant database access mechanism. However, a minority of applications (or application components) should be specified in a persistent programming language. It is an open challenge to provide both efficient SQL and efficient persistent programs in the same system, especially when the environment is update-intensive.

3.5. Integration of Structured and Semistructured Data

The advent of XML is likely to create an enormous quantity of data whose form is hierarchical rather than relational or object-oriented. Moreover, this is "semistructured," in the sense that many different forms of Web pages can fit a single schema. In spite of vigorous recent activity by the database community on query languages and environments for such data, the area is still in its infancy.

Database researchers have proposed declarative query languages for XML. However, given industrial activity in the area, we guess that XML and its evolving data manipulation languages will resemble a traditional hierarchical database system with a procedural data access language. The database research community should undertake the substantial effort of unifying web and database technologies, including the challenge of making web environments semantically appealing. Representative issues include handling sets of disparate, self-describing, potentially deeply nested objects, developing suitable declarative languages, loosely consistent transaction models, automated resolution, versioning, and interactions between updatability and caching.

4. The Grand Challenge

We propose that there be a grand vision that the research community attempts to accomplish in the next decade. The grand challenge should encompass most of the problems discussed in Section 3, but focus them on an important goal. People outside the field should find the goal easy to understand and exciting. Other fields have used such grand challenges to focus and motivate their fields.

We recommend a ten-year goal for the database research community:

> **The Information Utility: Make it easy for everyone to store, organize, access, and analyze the majority of human information online.**

The majority of human information will be on the Web in ten years. It will be an exabyte spread across the planet in many formats. Absent new tools, finding and understanding answers to our questions will be even harder than it is today. An ideal system would answer questions succinctly and would anticipate questions by notifying us of interesting events.

In other words, the goal is to turn the Web into a more useful information utility over the next decade.

5. Research Infrastructure

To encourage innovative work in pioneering areas, and specifically to accelerate the changes of emphasis advocated in this report, we recommend changing the reward system for researchers. Program committees of major conferences and journals should change their method of selecting articles. Also, electronic publication of technical reports has changed the way scientific literature should be collected and disseminated. We suggest changing the processes of the database system research community as follows.

First, CoRR and individual web sites provide an efficient electronic publication system [http://xxx.lanl.gov/archive/cs]. Conferences and journals should de-emphasize paper copies of their proceedings – rather they should present web sites that organize submissions and present editorial comments on them. We suggest that conferences move to an "all poster" or "mostly poster" presentation scheme. Many articles are so specialized that attendance at the presentation is limited to a few specialists interested in the topic. These specialists can have a more efficient group discussions at a poster session. Presentation slots should be allocated to ideas that do not follow the "delta-X" mentality, and to invited presentations that summarize recent progress in established fields and innovations in new fields.

In addition, we believe that information dissemination is best accomplished by accepting a substantially larger number of articles. This would make room for more innovative articles, without crowding out the strong delta-X results that conference reviewing tends to favor.

Lastly, and perhaps most controversially, we propose a public reviewing process. Every program committee and editorial board goes to a great deal of work to review a large collection of submissions. We want to somehow capture and publish this valuable information. Once an author makes a document public, e.g., by submitting it to CoRR, volunteer reviewers should be able to publish their reviews in a moderated forum. Organized reviews of related articles that compare, and contrast the material will be especially useful. We endorse the efforts of H. V. Jagadish to explore these issues and start such a reviews database.