AMERICA240

Designing the *B-1*

A Universal System for Information

Michael J. Carey[*], Joseph M. Hellerstein[§], Michael Stonebraker
{carey, jmh, mike}@cs.berkeley.edu

Abstract

Today's information systems are based on designs that are a quarter-century old: centralized servers with local metadata, black-box user interactions, complex monolithic software, and underlying hardware assumptions frozen at the date of initial design. Industrial system developers are firmly mired (by necessity) in this legacy of assumptions and historical arcana. Database researchers need not be.

We are beginning anew, designing the *B-1* (pronounced "be one") universal system for information. In building the B-1, we intend to explore a number of new ideas, combining them with suggestive features of recent research, as well as including a considerate exploration of roads not taken years ago. Our design goal is nothing less than a truly universal system for information: we envision a global federation of information stores inter-operating seamlessly and autonomously, removing today's artificial distinctions between databases, file systems, and the world-wide web, while simultaneously acknowledging and addressing the complexity of searching, browsing and analyzing the world's information.

This paper presents our vision for the B-1 system. As motivation, we point out scenarios where current information systems provide insufficiently powerful interaction, and we highlight mechanisms both to capture the attention of a broad community of users, and enhance their ability to make use of information. We also observe trends in hardware that suggest significantly different software solutions. The B-1 addresses both of these themes with an architecture of federated components. We advocate simplicity in the software, favoring flexibility over absolute performance and implicit feedback over explicit mechanisms. We attempt to strategically locate the complex components of the software in the large federation infrastructure to minimize the difficulty of engineering, adopting, and maintaining the local stores in the system. Our goal in sharing the B-1 vision at this formative stage is to invite and perhaps inspire other database researchers to join us in trying to make this vision a reality.

[*] Work done as a Visiting Fellow at UC Berkeley, Spring 1999; author's permanent address is carey@almaden.ibm.com.

[§] Contact author: jmh@cs.berkeley.edu, 387 Soda Hall #1776, Berkeley, CA, 94720-1776. 510/643-4011. Hellerstein and Carey are both on the VLDB program committee.

Designing the *B-1*

A Universal System for Information

Michael J. Carey[1], Joseph M. Hellerstein, Michael Stonebraker
{carey, jmh, mike}@cs.berkeley.edu

Abstract

Today's information systems are based on designs that are a quarter-century old: centralized servers with local metadata, black-box user interactions, complex monolithic software, and underlying hardware assumptions frozen at the date of initial design. Industrial system developers are firmly mired (by necessity) in this legacy of assumptions and historical arcana. Database researchers need not be.

We are beginning anew, designing the *B-1* (pronounced "be one") universal system for information. In building the B-1, we intend to explore a number of new ideas, combining them with suggestive features of recent research, as well as including a considerate exploration of roads not taken years ago. Our design goal is nothing less than a truly universal system for information: we envision a global federation of information stores inter-operating seamlessly and autonomously, removing today's artificial distinctions between databases, file systems, and the world-wide web, while simultaneously acknowledging and addressing the complexity of searching, browsing and analyzing the world's information.

This paper presents our vision for the B-1 system. As motivation, we point out scenarios where current information systems provide insufficiently powerful interaction, and we highlight mechanisms both to capture the attention of a broad community of users, and enhance their ability to make use of information. We also observe trends in hardware that suggest significantly different software solutions. The B-1 addresses both of these themes with an architecture of federated components. We advocate simplicity in the software, favoring flexibility over absolute performance and implicit feedback over explicit mechanisms. We attempt to strategically locate the complex components of the software in the large federation infrastructure to minimize the difficulty of engineering, adopting, and maintaining the local stores in the system. Our goal in sharing the B-1 vision at this formative stage is to invite and perhaps inspire other database researchers to join us in trying to make this vision a reality.

1 The Case for Starting Over

The computing world has changed dramatically since relational databases were first introduced in the 1970's. Despite the changes around them, however, the general architecture of today's commercial database systems is remarkably similar to that of the first relational database systems. Consequently, today's systems are straining to solve today's problems, and we believe that they are ill-suited for the some of the important challenges that lie ahead (particularly in the brave new world of the world wide web). Rather than patch and extend the initial relational designs further, we are undertaking the design of the *B-1*, a new universal system for information. Before describing our vision for the B-1, we highlight the reasons why we believe today's systems will prove to be insufficient in the long run, and we present our view of the key design requirements for the next generation of information technology.

1.1 Changes in Hardware

The most obvious changes in the last quarter-century have been in hardware; these changes alone are radical enough to consider rethinking database storage architectures from the ground up. Consider the following trends in disk performance [Patt99]:

- Capacity growing at 60% per year (2x / 1.5 years)

[1] Work done as a Visiting Fellow at UC Berkeley, Spring 1999; author's permanent address is carey@almaden.ibm.com.

- Mb/$ growing 100% per year (2x / year)
- Transfer rate (bandwidth) growing at 40% per year (2x/2.0 years)
- Seek/rotation time (latency) shrinking at only 8% per year (1/2 in 10 years)

One foregone conclusion from these trends is that storage systems will continue to grow very quickly, as their price and capacity are increasing rapidly. Since RAM sizes are also growing at a rate of 60% per year, we should soon expect to see terabytes of RAM serving as a cache for petabytes of disk. Another important trend is the ratio between disk latency and bandwidth. At its peak performance, a database system is transferring data at the maximum bandwidth of its disks. Performance potential is wasted during a seek since a disk arm is not delivering data during that time. We refer to the *bandwidth potential* wasted by a seek as the number of bytes that could have been transferred by sequential I/O in the time taken to do a seek. The bandwidth potential of a seek increases as the product of the bandwidth increase and seek overhead increase, i.e. it is growing at a factor of 1.4 * 0.92 = 1.288, or about 29% per year (!).

These two factors, the increase in RAM sizes and in wasted bandwidth potential, are conspiring to make current systems' 4KB and 8KB page sizes look increasingly foolish. Dividing a terabyte of RAM into 8KB pages results in over 100 million pages to be managed in memory – an absurd overhead in per-page metadata in memory. Moreover, the bandwidth potential lost during a seek is not nearly amortized by 8K page transfers. These phenomena are being aggravated at an exponential rate, and the time has come for significantly larger units of transfer.

Another ramification of the increasing cost of seeks is that "arm-wasting" architectures like RAID-5 are becoming increasingly unattractive. When the bandwidth potential wasted during a seek is high, it is particularly detrimental to have all the disk arms in a system seek in lock step. On the contrary, it is desirable to use schemes that explicitly stagger seeks and transfers across disk arms. The goal, in order to ensure that the system attains its full performance potential, is to have steady-state behavior in which the bandwidth of the disks that are *not seeking* equals the aggregate bandwidth of the I/O path through the system (including interconnection network capacity and CPU overhead per byte).

1.2 Application Architectures and Requirements

Relational database systems were designed to serve as large, unified solutions to the problem of enterprise data management. However, they are almost never used that way. Most organizations today employ database systems in a three-tier architecture, with clients at the top, TP monitors and packaged applications running code in middleware, and database systems at the bottom. It is by no means clear that this architecture is desirable: the cost of all this software is high, and maintaining both a database system and a TP monitor is difficult and expensive. In addition, the performance of this architecture is often sub-standard: it is typically preferable to run the code close to the data rather than in middleware. In addition, many three-tier architectures employ middle-tier data caching for performance reasons, leading to potentially thorny cache vs. database data consistency problems. Until recently, TP monitors were required to unify the operation of code; today, object-relational systems help solve this problem by providing support for user-defined functions, procedures, and methods.

With the global reach provided by web access, most modern enterprises require serious 24x7 operation. However, today's relational and object-relational database systems generally do not provide sufficient availability on their own.

Often the 24x7 availability requirement is quite stringent, and must be met even in the face of natural disasters and bursty seasonal workloads. A variety of inelegant, expensive patches exist for working around these problems today – for example, replication servers can be used to maintain warm standbys across wide-area networks to handle natural disasters, and over-engineering a system can solve bursty workloads by paying for peak performance up front. However, these techniques are neither efficient nor easy to maintain. Solutions to these issues should be designed into the architecture of the DBMS, not "bolted on" ex post facto.

1.3 Disunity in Storage Software: The Rise of the Web

Essentially all organizations are now storing important information in a hodgepodge of file systems, web servers, and database systems; this is the case despite the fact that, at a concrete architectural level, these systems do many of the same things. It is our belief that the divergence of file systems and databases was a historical error that has been radically exacerbated both by the popularity of the web as well as by the rise of recent "one-off" storage systems built for simple tasks like e-mail and calendars. Rather than argue about whether the database should be layered over the file system [Ston81] or vice versa [Ols92,CDF+94], we believe that these systems should be unified into a single storage layer that is shared for all modes of access.

For purposes of wide-scale adoption, it is clear that tomorrow's storage servers must have a web-centric architecture. This means that they must export HTTP as a standard interface. Unfortunately, HTTP 1.0 is not a client-server protocol. It is stateless, which makes it unnatural for the transactional connections that are expected in database systems. A variety of inelegant solutions are currently used to work around this, including the dynamic creation of URLs and the use of cookies stored in a filesystem. An additional issue in the rise of the web is the emergence of formats like HTML and XML; their loosely structured nature strains the capabilities of today's query languages and rigidly structured data models.

1.4 Scalability Requirements

Enterprise Resource Planning (ERP) and e-commerce applications are now requiring scalability that was previously unheard of. It is no longer uncommon to expect 100,000 simultaneous connections to a single system from around the globe. Future information systems must be able to handle workloads of this scale gracefully, balancing load and shepherding resources economically during peak- and off-season periods.

In addition to having more and more users connected, we can expect more and more data to be made available. Data collection technology is far outstripping data storage and analysis technology. For the foreseeable future, we expect the amount of data to continue growing as fast as storage technologies will allow. One typical scenario that is already emerging is to analyze large volumes of operational data with "big picture" decision support and data mining applications. As databases grow in size, the typical query will essentially end up running "forever" – that is, even a simple scan of "all the data" will take too long to be useful for most users. As a result, sampling and approximation techniques will become increasingly important. Again, we believe from experience that adding these techniques to existing systems is an expensive and inelegant solution. In the next-generation of information systems, sampling should be designed in, not bolted on.

1.5 Languages and User Interfaces

Enormous volumes of data are making it increasingly difficult for people to understand both how to find information and how to interpret the information they find. This is especially true as people attempt to integrate well-organized data with text and other loosely structured data sources. The increasing volume of information available for searching raises basic issues in human-computer interaction. Currently, different systems strike different balances of responsibility between user and computer. SQL-based systems place most of the burden on users and application developers, expecting them to carefully design their databases before loading, and then to query them precisely. In principle, SQL users get what they ask for – but it can be very difficult to figure out exactly what to ask. Text-search systems, in contrast, assign very little responsibility to their users. Textual documents can be entered into the system as natural free-form structures, and they can be queried with relatively free-form natural language queries. The problem here is that the computer is then expected to provide the thinking that users omitted, and the result is that users typically do not get what they are really asking for because there is no well-defined query language or data model to reason with.

In both of these frameworks, users are currently forced to iterate with the system frequently, posing and refining queries until they find what they believe they are looking for. The difficulty of this process is exacerbated by the long "think times" of many queries – if a user asks an inappropriate query, the system may not respond for a long period of time. The language-centric, request/response model of querying is essentially being pushed to the breaking point by the complexity of today's data. In the next system, an appropriate infrastructure for intra-query user interaction should be built in, enabling users to guide processing and stop early when answers look either sufficient or unpromising. Significant consideration must be given to the needs of data visualization tools and to the user-interface mechanisms for interacting with queries. The responsibility of tomorrow's systems simply cannot stop with the presentation of a result-set cursor.

1.6 Heterogeneity of Hardware

We are already in a world of very heterogeneous hardware, and we expect this trend to continue. Enterprises today use a mixture of clusters, SMPs and NUMA machines. Currently, most database vendors maintain at least 2 separate code lines for different architectures. We expect the heterogeneity problem to become exacerbated as workloads become bigger and networks become faster. It will soon be economically irresistible to construct systems that connect hundreds or thousands of heterogeneous machines into a cluster of sub-clusters. In order to remain cost-effective, customers will want to be able to grow these clusters incrementally, buying the latest hardware as necessary to meet the current level of user demand rather than being forced into buying today's hardware in anticipation of tomorrow's needs.

Simultaneously, the "Post-PC Era" is resulting in the proliferation and movement of personal data managed in a variety of guises (smart cards, personal organizers, phones, pagers, and combinations thereof). This stretches the range of computing diversity even further. Currently, many database vendors are selling or developing yet another "lite" code line for personal organizers. Mobility also ensures that network heterogeneity will be rampant, even if satellites blanket the globe with connectivity – low-power devices will be quite loosely connected when they are connected at all, while tomorrow's local-area networks will run at the speed of today's memory busses. In addition to mobile devices, the use of databases in embedded devices is expected to rise in the next few years. While we may not expect to run SQL in every

lightbulb, it may not be unreasonable to expect to be able to query a home control system about the state of the sprinkler system, alarm system, air conditioning system, and so on.

We believe that a single information system – based on a single code line – should be able to scale both up and down. The proliferation of post-PC gizmos also suggests that support for loosely-connected operation should be built in, not bolted on.

1.7 Code and Data

As noted earlier, there are good performance reasons to unify code and data. Keeping code and data together can also simplify application development. We do not expect hands-down winners in the war of component standards. As a result, systems will need to support multiple interfaces: Corba, OLE, DataBlades, Extenders, etc. They will also need to support multiple languages: C++, Java, Visual Basic, and so on. Object-orientation features have proven valuable to customers, and features like inheritance can be expected to be "must-haves" into the future. Adding all these features to systems ex post facto is proving difficult for most vendors. Method support and facilities for managing complex data types should be the deployed state of the art right now, but they are not; this is due in large part to the difficulty of teaching new tricks to aging code bases. It is much easier to have these features built in rather than bolted on after the fact.

1.8 Software Complexity

Researchers are often frustrated when vendors have insufficient interest in adopting their new technologies [CS99]. This problem arises in part because all vendors are having a hard time debugging their next release. The software base of industrial database systems has become bloated with decades of optimizations and bolted-on features, including stored procedures, multi-language extensibility, triggers, and of course an enormous pile of benchmark-improving hacks. At the same time, database users remain highly intolerant of errors; database systems are typically held to a much higher standard than any other software – certainly higher than the operating systems on which they run! (How many customers insist on recoverable file systems, or expect file-based applications to continue working across crashes?) Database vendors are crying out for magic bullet solutions to their coding problems, even to the extent of encouraging academic computer science departments to hire software engineering faculty.

It is not clear that software engineering research (or all the King's horses) can rescue system vendors from themselves at this juncture. A more reasonable expectation is that better engineering practices can be applied in the construction of a new system than were applied over the last 20 years in the construction of the systems in use today.

1.9 What's a Database Researcher to Do?

We have outlined many changes in the world in which database systems live. Section 2 will translate these changes into requirements for the new generation of information systems. Section 3 outlines the usage model that we expect will arise in a truly global information system. In Sections 4 through 6 we discuss design issues for the three main modules of the *B-1*. Section 7 will close the paper by reflecting briefly on the project's goals and challenges.

2 Requirements for a New Generation

It is our opinion that this is a time of opportunity, probably more so than any other time since the mid-1970's. Industrial developers cannot afford to step up to the challenge of starting over at this point, yet the software community is clearly going to end up in serious trouble if everyone continues on a path of incremental evolution. We intend to step into this vacuum over the next few years and attempt to design and build a new system to tackle these problems. In this section we list what we feel are the most important design requirements for the new generation of information systems.

2.1 The Six Key Requirements

Requirement 1: Universal Storage.
The historical differences between file systems and databases can and must be bridged: there should *be one storage system*, handling HTTP, file system, and query processing requests. This storage system must be designed to account for current and future hardware trends, and it must be designed to scale effectively both up to large parallel systems and down to embedded and mobile devices without requiring the implementation and maintenance of multiple code lines.

Requirement 2: Universal Access.
At a higher level, there should *be one logical space* for all querying, analysis and navigation of information. Codd's vision of data independence should be taken to its natural conclusion: despite a variety of data formats, layouts, and geographic locations, a user's view of data should be based on the unified totality of all information available worldwide. Different users will have different views for the usual reasons of privacy and convenience, with a global abstraction, albeit loosely organized, to unite these views. Tools and infrastructure must be available for content providers to advertise and integrate new data into the global view.

Requirement 3: Global Scale.
An essential aspect of unifying data into a single space is that users should be able to access any data anywhere on the globe, at any time. The system must have truly global scale, allowing for the management of global resources, and handling a global community of users (some of whom will be adversarial). By definition, there are no "off hours" in a global system, meaning that the system must run 24x7; it must also be prepared to cope with the bursty usage that arises in globally-accessible systems. In order to guarantee efficient access to geographically distributed users, flexible data placement and replication must be built in without requiring global coordination of database administrators.

Requirement 4: Simplicity of Code.
The system architecture must isolate its complexity into centrally-administered components. It seems impossible to architect a system with no significant complexity, but if the necessary complexity is isolated, it will at least be more manageable. For example, from a usage perspective it may be reasonable for a small number of service providers to administer and maintain complex software in the computing infrastructure. On the other hand, the storage system should *not* be centrally administered. Rather, it should be ubiquitous, running on a variety of platforms, and should be largely if not completely self-administering and self-tuning.

Requirement 5: Unification of Application Code and Data.
There should *be one repository for application code, data, and metadata*. However, code management needs to be as global as the system itself. There must be support for multiple languages and multiple hardware platforms, and pieces of application code should move as fluidly around the globe as the associated data. The additional facilities currently provided by TP monitors – reliable messaging and load balancing of application code – should be provided by the system along with management of the application code objects.

Requirement 6: Interactive Interfaces
With a global-scale system, it will be too hard for users to pose the right questions *a priori* regardless of the cleanliness of schemas or languages. On the other hand, hypertext-style or file-system browsing will be insufficient for users to find what they need in large volumes of information. A new system must integrate the notions of querying and browsing, and it must serve up incremental results and support incremental query reformulation.

These six requirements have led us to consider a number of designs for the B-1. Before we go into the low-level details (to the extent that they exist) of our preliminary B-1 design, however, let us briefly consider some over-arching architectural considerations that arise when attempting to build a globally scalable system.

2.2 Federations, Economics, and Scalability

Fundamentally, uniting the world's information depends upon uniting the world's users and system administrators. Attempting to accomplish this in a centralized fashion is a hopelessly utopian goal. Major corporations have a difficult time even getting their own IT shops to coordinate on centralized data warehousing schemes: some database administrators flatly refuse to share their data (or even copies!) with colleagues without an executive order. As a result, the only feasible administrative architecture for a globally scalable system is that of *federation*, where computing and data resources are only loosely affiliated. The world's users and administrators will unite only when the benefits of affiliation outweigh the risks of exposure – both of resources and information. A key aspect of overcoming this risk is to leave autonomous control in the hands of the people maintaining the resources. Another key aspect is to give administrators and users economic incentives to cooperate; the changing face of the web has demonstrated yet again that money makes the world go around, even when the world is virtual.

Federation is not just an attractive social structure; it is the only feasible software architecture for global-scale systems. Centralized mechanisms for scheduling, protection, and resource allocation simply do not scale to the globe. Federations allow users and administrators to maintain local autonomy over their resources, affecting local scheduling directly, and affecting the behavior of others only indirectly. In a federation, the n^2 communication between all parties is carried out *implicitly*, as the transitive consequence of 1-to-1 arrangements made between pairs of mutually interested parties. This implicit control has the ability to scale essentially indefinitely. While the lack of explicit control over the entire system can result in occasional global inefficiencies, the benefits in scalability are undeniable.

While global optimality is essentially hopeless, efficiency can still be encouraged in a federated infrastructure. Economic computing federations – so-called *agoric* federations [MD88] – are able to moderate the behavior of users and administrators by providing incentives for rational behavior. In the case of information systems, economic incentives

encourage content providers to share their information, and they encourage information seekers to take advantage of information that is available. Economics also discourage users from abusing the system in irrational ways – when one has to pay to play, one usually plays carefully. Finally, economics encourage efficient solutions to load-balancing problems. If an enterprise anticipates peak-season overloads, it can choose to enhance the system to meet demand, or it can subcontract work to other resource providers, who may in turn charge a premium for such service during rush periods.

Agoric systems were proposed for the management of general distributed computing resources, but they appear especially promising in the context of global information systems. The resources being managed in an information system have a value that is extrinsic to the system – everyone cares about information, while very few people actually care that much about the efficient use of CPU cycles today. Database developers and researchers should naturally be at the forefront of agoric computing research, since we manage the most valuable resources in computing. (We cannot afford to succumb to agoraphobia!)

3 Global Information and Usage Models

The B-1 goal is to allow people to interact effectively with enormous amounts of federated information. This virtual information base will be overwhelmingly large and disorganized, and will have all the properties of reality: it will be alternately authoritative and misleading, coherent and contradictory, elusive and intrusive. Once available, however, people will surely want to take advantage of it. A system of this scale will inherently require new modes of access that better reflect how people interact with information in reality.

The challenges of interacting with information have been treated in a number of communities in computer science, but never at the scale we are suggesting here. In attempting to unify the worlds' data, we as database researchers must necessarily take on the union of problems that are currently being addressed with varying degrees of success by query languages, information retrieval, data visualization, and statistical or probabilistic methods. At base we are engineers, and believe in building tools to allow users to solve these problems. While we see value in statistical methods, we believe that the scales should tip towards well-structured languages and interfaces rather than relying on artificially intelligent techniques. Care will undoubtedly be required in choosing how much to automate; we intend to keep a close eye on this balance as we progress with the B-1 design.

3.1 Deficiencies of Current Models

Of course we intend for the B-1 to subsume prior art and to be usable in a manner identical to current SQL systems, file systems, and the web – these usage models have demonstrated utility on varying scales. In fact, we are eager to ensure that the B-1 supports these modes of interaction so that it will be widely adopted, particularly at the low end (i.e., where web servers and file systems reign today). However we view these modes of access as inherently brittle, and do not foresee them scaling over the long term into the typical modes of interaction with a global information base. We briefly consider the deficiencies of the current data representation and usage models for information systems.

Hierarchies and Hyperlinks: File systems and the web support hierarchical and networked data models, respectively, with data being fetched incrementally by name or by pointer. It has been repeatedly argued and demonstrated that hierarchical and networked models of information do not scale well: users become confused and applications become unmanageable when hierarchical and networked data representations are scaled up and maintained over time. ("Where did I put that file?" "Which book in the library has the information I need?" "Why is this program unable to navigate to the right object?") Add to this the lack of structure in the individual objects in the file system and web spaces today, and it is clear that we are heading for real trouble. File systems and the web both ignored Codd's lessons of data independence, and the resulting problems are increasingly evident. Of course, hierarchies and hyperlinks (and their incremental traversal) can be useful in certain contexts; we submit that these contexts are by nature user-specific, however, which motivates the need for independence of flexible hierarchical and networked *views* over better underlying data models.

There's Something about Queries (and it isn't pretty): While we are not enamored with web browsing, strict logical models of querying (e.g. SQL) have been pushed to the breaking point as well. Global information systems are so complex that users are unlikely to be able to generate well-formed meaningful queries. Even with today's isolated databases, users can generate dozens of pages of SQL in a single query [CS99]. Supposing that a user could succeed in posing a well-formed query over a global system (say via a graphical tool), the query's answer is likely to be misleading due to information that is either unavailable, unreliable, self-contradictory, or subject to debate. This difficulty is exacerbated by the "black box" batch interfaces provided by traditional databases. Strict querying is appropriate for trained users or carefully-crafted applications over well-manicured datasets; however, it is too restrictive for a global-scale information system.

3.2 Interactive Evidence Accumulation

While we foresee traditional modes of access being used on localized subsets of data, we believe that large-scale information search will be an interactive, iterative *evidence accumulation* process. Consider the query `"Are there any really good Italian restaurants within 15 miles of where I live?"` The answers may reside in multiple databases that are federated together. Each database may rate restaurants with different units ("4 forks", "3 stars"), some databases may have the text of reviews but no ratings, and the trustworthiness of a rating will vary both with the reviewer and with the perceptions of the person asking the query.

To begin searching the data, users will often favor a "big picture" summary result, which gives them a sense of the options for pursuing their information. In our example query, the system might quickly discover a number of restaurant databases and then begin to stream out Italian restaurants and their ratings. Rather than waiting for complete results from this wide-area query, users will expect CONTROL-style online behavior from the system [Hel97]; this is especially important in a massive global system, which could return moderately relevant records *ad nauseum* as it finds more and more data sources. Unlike the relatively static SQL-oriented interaction of online aggregation techniques, one can imagine a much more fluid scenario here. For example, as more data streams in, the grouping might initially be geographic, aggregating the results into regions on a map. Later, the user might interactively change the grouping specification, e.g. to group by a combination of price range and rating as well, rather than just geographic location. We

foresee a rich interplay between modes of interaction that include online query result reporting, graphical programming, clustering and classification, and data visualization. Like most good database problems, this involves unifying, extending and scaling a broad spectrum of techniques.

3.3 Flexible Information Modeling

Obviously, to federate information on a global scale, the B-1 type system will need to allow for aggregation of dissimilar objects. While we indeed expect dissimilar data (forks and stars, or apples and oranges) to be combined and rolled up together, we do not necessarily endorse the schema-less, graph-centric query models favored by the database community's advocates of semi-structured data. Because of the current state of the web, it has become fashionable as of late to take a pessimistic, least-common-denominator approach to integrating strongly and loosely typed data. Despite the current state of the web, however, we expect that a large percentage of the truly *valuable* information in the world will be reasonably well-typed. We believe this because of the fact that content providers – or independent information consolidators – will surely have economic incentives to publish reasonably well-manicured data. One financial incentive comes from serious end users. (What would you give if you could find the data you really wanted on the web?!?) Another comes from the potential for eliminating the administrative and consistency problems that many smaller enterprises face today; their operational inventory and customer information is database-resident, and therefore structured, while their web pages are separately maintained in HTML form and are often out of date with respect to the actual items in stock.

Given the economically-driven eventuality of reasonably well-typed information becoming available on the web, we believe that it is better to focus from the beginning on information modeling approaches that retain a notion of type and of partially predictable structures. The minority of unstructured data can be fuzzily promoted into a more structured type system (e.g., typed objects with exceptional attributes) on the fly rather than demoting all of the world's data up front into a global soup of attribute-value pairs. In our view, the database community has an opportunity, and an obligation, to lead rather than follow (e.g., by not simply accepting HTML, or even XML, as a given and inevitable data model) in bringing more structure to the global information system.

4 Preliminary System Architecture

As we have explained, we expect the B-1 to serve users in a manner that combines many of the features of the web and database systems, with an increased focus on interactivity and approximation. We now proceed to discuss our thoughts on the architecture of the system itself. At this stage, we have a vision for the B-1, but we are carefully avoiding being doctrinaire about specific implementation solutions. In each section, we will attempt to highlight our themes and then describe a technique (or sometimes several alternative techniques) for realizing the B-1 vision.

4.1 Overview

Broadly put, the B-1 will be a federated, shared-nothing database system, which should scale to a modest number (order 10^4) of thick *federators*, a larger number (order 10^7) of relatively thinner *query servers*, and a very large number (order 10^9) of thin *stores*. The federators will be loosely affiliated in a computational economy, and they will be made aware of each other's services (for both data and processing) via a system of *metadata servers* that will itself be loosely

federated. We expect that federators, query servers, and stores will run on their own dedicated (though off-the-shelf) hardware; put differently, we expect B-1 to run on B-1 *appliances*, not on general-purpose computers. This means that we will require only minimal services from an Operating System – in particular, device drivers, lightweight thread management, and network services. Other features, including inter-process protection and communication, file system support, and security infrastructures will be unnecessary, and would be more than likely to just get in the way. We now proceed by considering each of the B-1 components in detail, from the bottom up.

4.2 The B-1 Storage Server (BOSS): A Thin Store

Our goal in designing the B-1 Storage Server (BOSS) is to make it simple and efficient: it must be easy to implement and maintain from a software engineering perspective, and it must be trivial to use and administer from the user perspective. We expect one standard architecture to have a BOSS embedded inside the controller of an intelligent disk [KPH98], and thus it cannot require significant tuning – particularly if it is installed in a rack with 100s of other such devices. Thus, it must be largely, if not entirely, self-managing.

The functionality of a BOSS is akin to the storage component of a traditional database (a la the RSS of System R): it will be an (optionally) transactional storage manager supporting object fetch via ID, selection, projection, and primitives that can be employed for joins. The design, however, must diverge significantly from current systems to take into account the previously mentioned changes in hardware and the need for simplified code.

4.2.1 Disk Management

It is patently clear that the unit of transfer from disk must increase, to accommodate the growth of RAM, and more importantly to avoid the exponentially increasing bandwidth potential wasted during random I/Os. We believe that the time has come to reconsider an old idea: the *segmented store*, as typified by early systems like the Burroughs B-5000 (one inspiration for our system's name), or Multics. In a segmented store, the units of transfer are large, variable-size segments that represent a logically coherent set of data. In general, we expect that the segment size should be on the order of 1/100 to 1/1000 of RAM, to ensure a manageable amount of per-segment metadata in memory. Segments should be loadable with relatively low latency, to allow for interactive response times to small cold lookups. For traditional relational data, each segment may hold a horizontal fragment of a table that may or may not be described by a logical predicate.

Note that variable-sized segments complicate buffer management with concerns about defragmenting memory. Additionally, variable-sized segments require schemes for splitting and coalescing segments as they grow and shrink; work on this topic needs to be dusted off and reevaluated in light of today's technology parameters. On the positive side, variable-sized segments more naturally associate units of storage with logical objects (e.g., horizontal partitions of tables, large objects like files, etc.) They also naturally accommodate data compression on disk; decompressed pages are by nature variable sized, which causes complications for paginated systems [IW94]. Given the power of modern processors, compression seems to be a natural performance enhancer, and yet it has been only recently been considered by the database research community.

4.2.2 Access Methods

We are currently considering a number of designs for indexes in this environment. Clearly, paginated indexes like B-trees make no sense in this environment, since we do not expect to have a traditionally paginated store. Instead, we expect indexes to reflect large main-memory data structures, e.g. T-trees [LC86] or perhaps treaps [SA96]. Two reasonable options exist for maintaining indexes on disk. The first option is to store indexes along with the data in a segment, so that an entire main-memory sub-database is fetched and written in one seek. This is another motivation for making segments variable-length, since the number and sizes of indexes can vary over time.

The second option is to never write indexes to disk, but rather to construct partial indexes in memory on the fly as segments are fetched and used, much as today's object databases swizzle pointers on the fly. This has the advantage of removing much of the challenge from physical database design, as no indexes need be chosen and constructed up front at all. This fits our philosophy of simplifying the tools, rather than the *au courant* vision of bolting additional automated tuning logic onto already complex systems. A lack of stored indexes also removes the need for index-related logging, as indexes never need to be recovered. An additional advantage is that by not relying on persistent indexes, we reduce the opportunity for their bugs to cripple the system, as bad pointers can be sidestepped by dropping an index and reconstructing it. The ability to tolerate bugs – especially the tricky so-called "Heisenbugs" that often arise in concurrent systems [GR93] such as intricate index concurrency and recovery logic – is a significant advantage in quickly engineering large global-scale systems [FGC+97]. Finally, hot segments that remain in memory for significant amounts of time could get indexed more aggressively than cold segments. Of course, this scheme assumes excess CPU cycles, which may or may not make sense. The tradeoffs between materialization and re-computation (and by whom, e.g., in an intelligent disk environment) of secondary indexes will be an interesting aspect of the B-1 design.

4.2.3 Logging and Recovery

Current log managers present a software engineering nightmare. They are extremely complex pieces of code, which are very difficult to test, and do not get well exercised in the quality-assurance cycle. Moreover, when they do fail, the results are disastrous to the client, and can turn into embarrassing front-page news for the system’s designers.

One of our basic design goals is to simplify logging and recovery, as a means to lower the complexity of the BOSS. A traditional database log is essentially a versioned copy of the database, encoded in a different format than the database itself. Much of the challenge of logging arises from the efficient and correct translation of updates to log format, and the subsequent translation from log format to database format during recovery. In order to simplify this logic, the B-1 thin store will use a single, uniform representation of the data; there will be no distinction in representation between “database” and “log”, there will simply be data, and physical data replication will be used for reliability. Note that data replication is in increasingly common use for availability anyway (as discussed further in the next section), so this should unify two necessary services into a single piece of code.

While we believe that the BOSS must have a single storage representation, we have not agreed on what that representation should be. Three basic possibilities present themselves:

1. **Log Only:** Each segment is stored as a log of the actions on that segment, with updates appended to the end, perhaps in a non-differential manner. Segments are read into memory backwards. Hot segments are translated into main-memory sub-databases containing the latest version of each object. Cold segments are streamed through memory on demand, with the latest versions of the relevant objects extracted for the request. This latter mode of access is also natural for modeling streaming data from sensors or multimedia sources. Outdated and aborted versions can be removed from segments in memory in either case, and the segments can be rewritten to disk as the opportunity presents itself.
2. **Versioned Database:** Each segment is stored in the same format that is used in main memory, and versions are kept in the segment for transactional purposes. This bears some similarity to the various multiversion concurrency control algorithms that have been proposed over the years (e.g., as examined in [CM86]), including the scheme that Oracle uses to allow long-running reads to progress simultaneously with writes**.** It also harkens back to the POSTGRES no-overwrite storage system [Ston87]. While the POSTGRES storage system had performance problems, the success of Oracle's scheme suggests that the problems were artifacts of the implementation and not inherent in the concept.
3. **Unversioned Database:** Each segment is stored in the same format that is used in main memory, but only the latest version of each object is kept. Note that replication can provide REDO logging, but the lack of versions means that there is no UNDO logging. Hence writing dirty segments to disk before commit violates ACID semantics. Unfortunately, a "NO STEAL" policy that never flushes dirty segments to disk is not viable, since memory will fill up quickly under such a scheme, resulting in low throughput. This scheme thus requires that all applications provide user-level compensation techniques to allow for rollback (or that the system resort to some sort of logging in order to do so itself, e.g., see [LC87]).

Note that recovery is trivial in all these schemes due to the single representation: a replica of the database is fetched from another location, and directly installed. Similarly, there is no need for checkpoints, since there is no "roll-forward" to speed up.

4.2.4 Replication

Every serious database installation requires 24x7 availability; this need will increase as computing becomes ever more interwoven into the fabric of everyday life. True 24x7 availability means that the system stays available in the face of disasters like power failure, operator error, and Acts of God. RAID and related "in the box" techniques provide no reliability to this kind of failure. Instead, every enterprise that truly values 24x7 availability currently uses a replication server to maintain warm standbys placed in a different geographic location across a WAN.

A variety of replication products exist in the marketplace today, and there are various approaches to implementing replication. One natural approach is to periodically copy the database directly; this is sometimes referred to as *snapshot replication* (or a "dump" in file systems terminology). Snapshot replication is typically inefficient: it tracks physical changes to the database rather than logical changes, and it typically does not allow for differential schemes, so old data is replicated multiple times. By contrast, replication tools like Sybase Replication Server [Sybase] or IBM Data Propagator [Gold95] do not extract data directly from data sources. Instead, they parse a DBMS log, convert committed transactions into SQL commands, and ship those commands across the network to a remote site. This *transaction-copying* implementation has certain advantages over snapshot replication. First, it can result in less data movement than data-

copying, since only the changes are propagated, not all the data. Second, transaction-copying ensures that only transactional data is replicated, not the results of in-flight or aborted transactions.[2] On the other hand, the transactions do have to be "replayed" at the destination, whereas snapshot replication can use bulk-loading facilities to improve the load performance.

The BOSS replication engine will be used not only for availability, but also for performance (data dissemination and caching) and for transaction rollback. Rather than having a separate database and log at each site, we expect that users with valuable data augment their database with a local "hot standby", and a remote "warm standby", both managed by the replication engine. This will serve the purposes of transaction rollback and availability respectively. Additionally, for purposes of performance, it is often useful to keep copies of data co-located with the users who query that data. Traditional client-server caching schemes had this in mind [FCL97], but they were not architected to scale to the globe. Mariposa's initial ideas on *data brokering* were not pursued in detail [SAS+96], but should be revisited, as we believe that an agoric model could prove useful for negotiating the system-managed placement and refreshment of copies. We note that some copies may be more stale than others, and that the timeliness of data should be taken into account as part of query specification and/or result reporting.

There is a great deal of hubbub these days about mobile computing and disconnected operation. It is not clear whether any device will ever remain totally disconnected for long in the future, but it is plausible to assume that some devices will go through periods of lower connectivity. The BOSS replication engine can be responsible for coordinating the placement of replicas onto and off of mobile devices, much as it is with connected devices. The trickiest aspect of aggressive replication – especially onto loosely-connected nodes – is the way in which distributed transactions are managed. Two-phase commit techniques are not an option; in fact, many distributed applications do not use two-phase commit even today, because of its high overhead. Lighter-weight, flexible solutions will need to be applied in the B-1 regime.

We close our discussion of replication by noting a tight interplay between the designs of the BOSS replication engine and the disk layout. If the single format in the BOSS is a log, then the replication engine can be implemented much like today's high-performance replication engines (with the exception that it need not parse the log, it need only ship the tail of the log and append it to the destination). In contrast, if the single format in the BOSS is a database format, the replication engine may be forced to resort either to snapshot replication or to the use of triggers.

4.2.5 Sessions, Users, and Security

In order for B-1 to become a universal system, it must be able to gain a foothold among today's users. As a result, we intend for it to be backward-compatible with both web and filesystem interfaces. As part of this, we intend to use HTTP as our basic communication protocol. In fact, the connection manager of B-1 will be a web server. HTTP 1.0 is a stateless protocol, so some mechanisms will be required to simulate statefulness of connections. Current schemes based

[2] Note that most data-copying mechanisms do not extract data using degree-3 consistency (two-phase locking), since they run for a long period of time and touch large portions of the database. In degree 3, this would lock out writers for the time of the extraction.

on impenetrable URLs or cookies are rather awkward. New versions of HTTP may help, but generally this is an area that needs to be explored further.

Today, every database system tends to have its own access control information on users, groups, roles and so on. Federations typically multiplex a single connection to a database: the federation software logs into an underlying database system as one user ("Federator"), and although multiple clients log into the federation, the federation uses its single "Federator" account in the underlying system for all the federation clients. This kind of connection multiplexing is critical for both performance and administrative scalability – the alternative is for all systems to recognize all users! Multiplexing connections raises a number of questions about authentication and other security issues. We believe that encryption mechanisms can play a major role as the basic session management and global user authentication scheme. If encryption is done right, there is no need for access control anywhere, meaning that underlying systems need not change much to participate in a secure global federation. Key management becomes tricky however – the system must ensure that the right people have the keys to the right objects. The management of distributed capabilities (keys) surely sounds like a distributed database problem, and merits further investigation.

5 The B-1 Query Server (BOQS)

Despite new usage models, we believe that a language akin to SQL99 or OQL will be a sufficiently powerful to serve as the command language for data processing in B-1. As a result, we focus here on the parallel, distributed evaluation of SQL-style queries in B-1; this will be carried out by the B-1 Query Server (BOQS).

We envision BOQS as an agoric, federated query processing system with an open API for fetching data from external sources. While we expect the BOQS to run efficiently over the BOSS, we also expect it to interoperate with third–party data sources via database connectivity protocols like JDBC. In a large federation, the basic assumption of traditional query optimization does not hold: execution costs cannot be estimated *a priori*. This is because costs change with resource utilization, resource utilization in a wide-area environment is constantly changing, and many of the system components are only semi-cooperative black boxes. Moreover, incremental query results make the resource requirements of a query unpredictable: a user may stop it early, or let it run indefinitely (especially if it queries a streaming data source like a television feed or chat room). Finally, the ability for users to modify queries on the fly turns the initial optimization of a query into a speculative exercise at best.

In B-1, we assume that resource utilization will change often, both between queries and during the course of a single query. We also expect that user specifications will change even within a single query. Therefore our general scheme for query execution will have the following general pattern:

1. **Initial Plan:** Find an initial plan via some distributed query planning scheme

2. Trace and Modify: During execution, performance of the plan is traced to highlight failures in expected performance, and opportunities for improvement. In essence, query execution is simultaneously a *feasability study* for the current query plan. Based on this tracing process, the plan can be changed as it runs.

5.1 Choosing an Initial Plan

This broad scheme leaves room for a number of design options. To begin with, the scheme used in the initial planning phase is unspecified. Because this is a large federation, traditional distributed query optimization is inappropriate: it does not account for changing resource utilization (and hence changing costs for operations), and it attempts to exhaustively search too large a plan space. Instead, we favor an autonomy-preserving optimization scheme like that of Mariposa [SAL+96] (which allows sites to "bid", i.e. to dynamically specify a cost for an operation) or Garlic [Haas 97] (which essentially allows sites to do their own costing for subplans).

Constraining the plan space is another key issue for query processing in a large information system context. A number of options are available to reduce the plan space considered. Mariposa constrained the plan space via a two-phase optimization similar to that used in parallel systems [HS91]: it first constructed a query plan based on traditional single-site query optimization [SAC+79] and then fragmented that plan via an economic bidding process. Both top-down and bottom-up bidding schemes would be possible and worth considering for B-1. Other heuristics for constraining the plan space include randomized algorithms [IK90] and job scheduling techniques based on constrained cost models [IK84,KBZ86]. Note that the technique used to search the plan space is orthogonal to the economic model used for costing. The only consideration that the economic model places on the search algorithm is the higher cost of costing: discovering the cost of an operation can involve contacting a remote node for a bid, which can itself be time consuming. As a result, schemes that repeatedly estimate costs of large plans (such as randomized techniques) may run slowly unless clever schemes are used for memoizing bids as they are received.

The models used for bidding and minimizing costs in Mariposa did not take into account the possibility of CONTROL-style online execution. Mariposa assumed that each query would be accompanied with a *bid curve*, which specified for each unit of time the amount a user was willing to pay for an answer to their query. If the optimizer could not find a plan that "fit under the bid curve", it would notify the user and reject the query. Online query processing complicates this picture with a third dimension, namely the quality of answers. One can imagine a query accompanied by a 3-dimensional "bid surface" of (time, quality, monetary value). However, it is not clear where such curves would come from in practice.

We recall in passing here that different sources will have different capabilities, and these capabilities can change with time. For example a system administrator of a participating SQL system may allow only index lookups during business hours, index lookups and table scans during the evening, and complete multi-table SQL access during nighttime. File systems may never support joins, and search engines may only support index lookups. This information must be captured by the optimizer, and factored into the constraints on the space of plans considered during optimization. These problems have been addressed in recent work [CHS+95,GHI+95].

5.2 Tracing and Reoptimization

In a large federated system, one must expect that performance will change frequently, often within the lifetime of a single query. We characterize these performance fluctuations as follows:

1. **Data sources:** The leaves of a query plan tree can have fluctuations in performance. This can arise due to performance changes at the data sources (e.g. because they become more or less loaded with other tasks), or due to changing network performance between the sources and the "data consumers" above them in the tree.

2. Query Operators: Intermediate nodes in a query plan tree can have fluctuations in performance as well, for similar reasons.

In the extreme fluctuation, a data source or query operator may simply cease functioning. If a data source becomes unavailable during processing, it may be possible to switch to a replica if one exists. If a query operator becomes unavailable during processing, the problem is more difficult, since its state may be more complex than a simple cursor over a table fragment. While we intend to consider issues of recovering in-flight queries, we do not consider this to be of paramount importance – the simple solution of restarting the query may often be the best solution anyhow.

A more likely scenario is for one operator in the tree to perform below or above expectations. When this happens, we see two possible alternative designs to compensate. At a coarse grain, one could rerun the economic optimization taking into account more recent prices. On a finer grain, our design could trade off the power of query operators for a more fluid system with continuous reoptimization. We plan to explore and compare these ideas.

5.2.1 Economic Reoptimization

In an economic model, forecasting can be used to predict – based on the current time and the amount of work left – whether the query will complete underneath the bid curve. When this is not the case, the system can reoptimize the query, and possibly reschedule some of the work. This is not unlike the approach of [KD98], extended into an economic model. Unfortunately, this design does not take into account the interactive nature of online query processing – in fact, the design of [KD98] allows reoptimization only after the completion of a blocking operator in a plan tree! A more CONTROL-centric approach is thus required for the B-1 to recognize a plan going over cost. One attractive scheme is to charge for query results much like a pay phone: after processing for a while, the system will ask the user (or an agent) for additional money to cover further processing on the given query. An alternative scheme is to avoid these issues via resource reservation, much as has been proposed for networks. This latter approach makes somewhat more sense for online processing than for the bursty resource utilization of blocking operators, but seems likely to under-utilize the available resources in the federation**.**

5.2.2 Continuous Reoptimization

A more radical option is to consider *continuous reoptimization*, with query plans built on non-blocking, *symmetric* operators – binary operators that do not distinguish between left and right inputs (e.g. merge joins, pipelining hash joins [WA91] and ripple joins [HH99].) The rough idea here is that the order of operators in a symmetric, non-blocking tree is easily changed on the fly, suggesting that the join order can be reset dynamically and automatically, per tuple of input from each relation. For example, when a table-scan operator *R* produces a new tuple, *any* join operator that involves *R* can take that tuple, and combine it with matching tuples from another relation. By iterating this idea, intermediate tuples can be dynamically "fleshed out" into complete result tuples, without specifying a join order a priori. This leads to a self-regulating, continuous reoptimization of the plan tree – the expected order in which tuples flow through the different operators can be determined by the rates at which the various producers of tuples can work, and the rates at which the

various operators can complete their tasks. Note that an "expensive route" through the operators will perform slowly, providing negative feedback that minimizes the number of tuples flowing along that route.

In simple cases like a single-site plan of merge joins, all the operators may have approximately the same overhead. However in a more complex plan with various kinds of ripple joins, as well as user-defined methods, this may not be the case. Moreover, in a parallel, distributed environment (with a distributed work queue, as in the River system at Berkeley [AAT+99]) the rates of progress may change during the lifetime of a query depending on network congestion, machine loads, etc. In essence, this allows the expected path of tuples to change dynamically over time. Viewed through the lens of traditional binary query trees, the system is *continuously reoptimizing* the tree – one output tuple may have traversed one tree, another a differently-ordered tree; on average, most tuples should get processed as if they were handled by the optimal non-blocking tree.

Symmetric, non-blocking joins are not necessarily the fastest algorithms available, but we believe that in many cases they will provide the best common-case performance, exactly because they are so adaptable; this adaptability is critical in large federated environments with unpredictable workloads. This payoff may not occur for small queries, however; good expected-case performance is not beneficial when only a few tuples are processed (since they may be "unlucky" and be treated suboptimally.) Thus we expect an initial, traditionally optimized plan to be of use to constrain the way tuples flow at first. As time progresses, the system can be allowed to begin reoptimizing continuously.

6 The B-1 Meta-federation Broker (BOMBer)

Any federated system needs metadata to describe the resources of the system. These resources include the machines and software modules that compose the system, and the data that these modules manage. In essence, the metadata is a compressed description of the entire federation. Typically the metadata in a federation will be highly replicated and somewhat fragmented, depending on access patterns.

In the B-1, we expect that multiple organizations will have an incentive to provide their own metadata; there will be a plurality of metadata providers. In essence, each metadata provider defines a federation, and hence B-1 is a federation of federations. The B-1 Meta-federation Broker (BOMBer) is a module that manages multiple sets of metadata, handles queries by querying the meta-federation, and generates more explicitly-defined SQL99-style queries on underlying BOQSes and BOSSes

6.1 Metadata, Discovery, and Crawlers

Traditionally, relational databases have used metadata for capturing both the logical and physical schema. An agoric system uses metadata not only to describe schemas, but also to allow sites to advertise their prices for certain operations. Advertising "pushes" price quotes to minimize the overhead of bidding during query optimization; it is essentially a broadcast-based replication scheme for physical metadata.

This federated metadata serves as a basic component of the system's operation. However we intend to leverage the BOMBer to also serve as a framework for information discovery. Using the same mechanisms, we expect large content providers to advertise and characterize their content via the BOMBer. Similarly, an intermediary service might run a

"schema crawler" that would issue catalog requests to databases on the network and load the resulting metadata – suitably transformed – into the BOMBer.

In essence, the metadata of B-1 will be very much like a Yellow Pages of businesses: sites can advertise their merchandise ("We have an extensive set of restaurant reviews, written by restaurant critics from leading newspapers"), and their services ("We can do 300 TPC-C queries per second; low prices!"). Many Yellow Pages providers may exist. Similarly, we expect data integration services to advertise themselves on the BOMBer ("We merge relational schemas, IMS and VSAM!"), resulting in an extensible system for fusing data sources. A plausible application might provide a user with multiple answers to a query, each integrated differently by multiple integrators. From an interface perspective this is similar to submitting a text-search query to search-engine aggregators on the web today (e.g. AskJeeves.com), but the scope and flexibility of sources and integrators in this scenario would of course require more sophistication.

7 Conclusions

In the last quarter century, information systems have entered the collective consciousness – they are deployed worldwide on numerous scales, from individual households to massive corporate IT installations. This has not happened in an organized fashion, and the software being used is uniformly insufficient for the various tasks at hand. Current trends in hardware, application architectures, web storage, scalability requirements, user interfaces, device heterogeneity, and software complexity and manageability are all putting significant pressures on today's aging information system architectures.

The current situation presents enormous opportunities and responsibilities for computer scientists, and for database researchers in particular. We believe this is a time of opportunity for researchers to start over, and design new information systems based on fundamentally different design choices than were made for today's systems. We see six requirements that we view as essential for the next generation of wide-area information systems: universal storage, universal access, global scale, code simplicity, code/data unification, and interactivity. We also believe that usage and information modeling requirements for such a system must be different from the state of the art. At Berkeley, we are trying to meet these challenges and requirements with the B-1, which we intend as a universal system for information.

Acknowledgments

The authors wish to thank the database research group at Berkeley for many fruitful discussions. Thanks also to Berkeley's OS and architecture groups for their willingness to build and cross bridges.

References

[AAT+99] Remzi Arpaci-Dusseau, Eric Anderson, Noah Treuhaft, David E. Culler, Joseph M. Hellerstein, David Patterson, and Kathy Yelick. "Cluster I/O with River: Making the Fast Case Common." To appear, IOPADS '99.

[CDF+94] Michael J. Carey, David J. DeWitt, Michael J. Franklin, Nancy E. Hall, Mark L. McAuliffe, Jeffrey F. Naughton, Daniel T. Schuh, Marvin H. Solomon, C. K. Tan, Odysseas G. Tsatalos, Seth J. White, and Michael J. Zwilling. "Shoring Up Persistent Applications". *Proceedings ACM SIGMOD*, Minneapolis, June, 1994.

[CHS+95] Michael J. Carey, Laura M. Haas, Peter M. Schwarz, Manish Arya, William F. Cody, Ronald Fagin, Myron Flickner, Allen Luniewski, Wayne Niblack, Dragutin Petkovic, Joachim Thomas II, John H. Williams, and Edward L. Wimmers. "Towards Heterogeneous Multimedia Information Systems: The Garlic Approach." *Proc. Fifth International Workshop on Research Issues in Data Engineering - Distributed Object Management (RIDE-DOM)*,1995, pp. 124-131

[CM86] Michael J. Carey and Waleed A. Muhanna: The Performance of Multiversion Concurrency Control Algorithms. *ACM Transactions on Computer Systems*, 4(4), November, 1986, pp. 338-378.

[CS99] Michael J. Carey and Len Seligman. "NSF Workshop on Industrial/Academic Cooperation in Database Systems." *SIGMOD Record*, 28(1), March 1999.
[FCL97] Michael J. Franklin, Michael J. Carey, and Miron Livny. "Transactional Client-Server Cache Consistency: Alternatives and Performance." *ACM Transactions on Database Systems* 22(3), September, 1997, pp 315-363.
[FGC+97] Armando Fox, Steve Gribble, Yatin Chawathe and Eric A. Brewer. "Cluster-Based Scalable Network Services". *Proceedings of SOSP '97*, St. Malo, France, October 1997.
[Gold95] Rob Goldring. "Update Replication: What Every Designer Should Know". *Info DB* 9(2) April, 1995. http://www.software.ibm.com/data/pubs/goldring/
[GHI+95] H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and Jennifer Widom. "Integrating and Accessing Heterogeneous Information Sources in TSIMMIS". In *Proceedings of the AAAI Symposium on Information Gathering*, Stanford, CA, March 1995, pp. 61-64.
[GR93] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, San Francisco, 1993.
[Hel97] Joseph M. Hellerstein. Online Processing Redux. *IEEE Data Engineering Bulletin* 20(3), September, 1997.
[HH99] Peter J. Haas and Joseph M. Hellerstein. "Ripple Joins for Online Aggregation." To appear, *Proc. ACM-SIGMOD International Conference on Management of Data*, Philadelphia, June, 1999.
[HS91] Wei Hong and Michael Stonebraker. "Optimization of Parallel Query Execution Plans in XPRS". *Proc. First International Conference on Parallel and Distributed Information Systems* (PDIS 1991). Miami Beach, December, 1991.
[IK84] Toshihide Ibaraki and Tiko Kameda. "On the Optimal Nesting Order for Computing N-Relational Joins." *ACM Transactions on Database Systems* 9(3), 1984, pp. 482-502.
[IK90] Yannis E. Ioannidis and Younkyung Cha Kang. "Randomized Algorithms for Optimizing Large Join Queries". *Proc. ACM-SIGMOD Conference on Management of Data*, 1990, pp. 312-321.
[IW94] Balakrishna R. Iyer, David Wilhite. "Data Compression Support in Databases." *Proc. 20th International Conference on Very Large Databases (VLDB)*, Santiago, Chile, September, 1994.
[KBZ86] Ravi Krishnamurthy, Haran Boral, Carlo Zaniolo: Optimization of Nonrecursive Queries. *Proc. 12th International Conference on Very Large Database Systems*, Kyoto, Japan, August, 1986.
[KD98] Navin Kabra and David J. DeWitt. "Efficient Mid-Query Re-Optimization of Sub-Optimal Query Execution Plans". *Proc. ACM-SIGMOD Conference on Management of Data*, Seattle, June, 1998, pp. 106-117.
[LC86] Tobin J. Lehman and Michael J. Carey. "A Study of Index Structures for Main Memory Database Management Systems". *Proc. 12th International Conference on Very Large Database Systems*, Kyoto, Japan, August, 1986.
[LC87] Tobin J. Lehman and Michael J. Carey. "A Recovery Algorithm for A High-Performance Memory-Resident Database System." *Proc. ACM SIGMOD Conference on Management of Data*, June, 1987, pp. 104-117.
[KPH98] Kimberly Keeton, David A. Patterson, and Joseph M. Hellerstein. "A Case for Intelligent Disks (IDISKs)." *SIGMOD Record* 27(3), 1998, pp. 42-52.
[MD88] M.S. Miller and K. E. Drexler. "Markets and Computation: Agoric Open Systems". In B.A. Huberman, ed. *The Ecology of Computation.* North-Holland, Amsterdam, 1988.
[Ols93] Michael A. Olson. "The design and implementation of the Inversion file system." *Proceedings of the Winter 1993 USENIX Conference*, San Diego, Jan. 1993.
[Patt99] David A. Patterson. Personal communication, February, 1999.
[SA96] R. Seidel and C. R. Aragon. "Randomized Search Trees." *Algorithmica*, 16:464-497, 1996.
[SAC+79] Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. "Access Path Selection in a Relational Database Management System". *Proc. ACM-SIGMOD Conference on Management of Data*, Boston, May, 1979, pp. 23-34.
[SAS+96] Jeff Sidell, Paul M. Aoki, Adam Sah, Carl Staelin, Michael Stonebraker, and Andrew Yu. "Data Replication in Mariposa." *Proc. IEEE International Conference on Data Engineering* 1996, pp. 485-494
[SAL+96] Michael Stonebraker, Paul M. Aoki, Witold Litwin, Avi Pfeffer, Adam Sah, Jeff Sidell, Carl Staelin and Andrew Yu. "Mariposa: A Wide-Area Distributed Database System." *VLDB Journal* 5(1), 1996, pp. 48-63.
[Ston81] Michael Stonebraker. "Operating System Support for Database Management." *Communications of the ACM*, 24(7), 1981, pp. 412-418.
[Ston87] Michael Stonebraker. "The Design of the POSTGRES Storage System." *Proc. 13th International Conference on Very Large Data Bases (VLDB)*, 1987, pp. 289-300.
[Sybase] Sybase Corp. "SYBASE Replication Server: A Practical Architecture for Distributing and Sharing Corporate Information." http://www.sybase.com/products/datamove/repserver_wpaper.html.
[WA91] Annita N. Wilschut, and Peter M. G. Apers: Dataflow Query Execution in a Parallel Main-Memory Environment. *Proceedings of the First International Conference on Parallel and Distributed Information Systems (PDIS)*, 1991, pp. 68-77.