An Agenda for Research in Large-Scale Distributed Data Repositories

M. Satyanarayanan
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Invited Paper for
Workshop on Operating Systems of the 90s and Beyond
Dagstuhl Castle, Germany, July 1991

Abstract

Access to shared data is provided today by distributed file systems and databases. In this paper, we explore certain usage and technological trends that will radically change the way shared data is used in the future. The usage trends include the growing need to access shared data from anywhere, increasing scale, and the increasing importance of efficient search. The technology trends include the advent of portable machines, the availability of software and hardware for using diverse types of data, and the growing diversity of network speeds and capabilities. These trends induce fundamental research problems in the areas of *adaptive system behavior*, *secure remote execution* and *extensibility*.

This work has been supported by the Defense Advanced Research Projects Agency (Avionics Lab, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U.S. Air Force, Wright-Patterson AFB, Ohio, 45433-6543 under Contract F33615-90-C-1465, ARPA Order No. 7597), the National Science Foundation (PYI Award), and the IBM Corporation (Research Initiation Grant). The views and conclusion expressed in this paper are those of the author, and should not be interpreted as those of the funding agencies or Carnegie Mellon University.

1. Introduction

A *distributed data repository* enables nodes in a distributed system to share information. An ideal instance of such a mechanism would be *transparent, efficient* and *ubiquitous,* allowing unobtrusive and fast access to shared data from anywhere. It would be *scalable,* allowing graceful expansion in terms of number of users, volume of data stored, and geographical area spanned. Even at large scale, the mechanism would be *secure, reliable* and *available.* Finally, the mechanism would be *flexible,* placing minimal constraints on the kind of information being shared, on the patterns of access to it, and on the data model presented to application programs.

The feasibility of distributed data repositories was explored in the 1980s by many projects in industry and academia. Distributed file systems, such as AFS [7] and NFS [5], and distributed databases, such as Ingres [9], are examples of present-day distributed data repositories. But although such systems are of great utility and commercial significance, they fall far short of the ideal. Research prototypes such as Coda [6], Echo [3], Starburst [4], and Postgres [10] come closer to the ideal in some respects. However, no system in operation today simultaneously addresses a number of emerging technological and usage trends. Some of these trends are already evident, and all of them are certain to be increasingly important over the next decade.

In this paper, we identify these trends and explore their implications for the distributed data repositories of the future. By examining these implications in the context of the current state of the art, we induce three key research problems: *adaptive system behavior, secure remote computation,* and *extensibility.* Substantial progress in these research areas is essential if the full potential of distributed data repositories is to be realized.

2. Future Demands

Two kinds of forces are at work in shaping the evolution of distributed data repositories. First, the way in which humans and organizations use data repositories is changing, thereby placing new demands on the repositories. Second, advances in hardware and software technologies are providing new opportunities in areas hitherto considered infertile. Although these two kinds of forces are interdependent, they are sufficently distinct in character that we consider them separately here.

2.1. Usage Trends

2.1.1. *Accessing data from anywhere*

The ability to conduct business from any location is a necessity in an increasingly mobile world. As end-users become more sophisticated and make direct use of data in shared repositories, there will be increasing need to widen the *span of access* of those repositories. Not being able to access necessary data at all times will put one at a competitive disadvantage, and will be viewed as unacceptable. Further, as the popularity of cellular telephones suggests, there is considerable latent demand for the ability to access data without being physically attached to a network.

A different, but equally compelling, forcing function is the increasing social acceptance of the home (or any other location) as a place of work. Although the traditional office environment is unlikely to disappear, there are many work-related activities that can be effectively done elsewhere. A CEO writing a speech for the next shareholders' meeting would probably be better inspired outdoors under a tree! If she could easily access relevant data, an investment banker with a young child could be productive even if she had to stay at home due to the absence of her babysitter.

2.1.2. *Increasing scale*

There is relentless pressure on successful distributed systems to grow, both in terms of number of users as well as number of nodes. It is easy to see why. As the volume of stored data in a shared repository grows, it becomes an increasingly valuable resource. At some point in the evolution of the system, access to this data is viewed as a necessity rather than a luxury by the user community. There is then considerable incentive to allow access to users who were originally outside the scope of the system.

Increased scale has performance, administrative and economic consequences. The negative impact of scale on performance is now well recognized. Algorithms and techniques that work well at small scale degenerate in non-obvious ways at large scale. Each quantum increase in scale exposes new ways in which the old tricks fail to work. Scale-related performance issues are certain to be an area of continuing research in the forseeable future.

Scale exacerbates the intrinsic tension between autonomy and interdependence in distributed system design. Whereas tight coupling between nodes tends to offer better consistency and transparency, weak coupling tends to offer better performance and availability. One possible compromise is to use a dynamic strategy as in Coda. Under normal conditions, Coda offers relatively tight coupling. But when failures make tight coupling impossible, Coda transparently falls back to a weaker coupling. While Coda has shown the viability of such a strategy for the Unix file system model, its use in a broader context remains to be explored.

A large distributed system is unwieldy to manage as a monolithic entity. For smooth and efficient operation it is essential to delegate administrative responsibility along lines that parallel institutional boundaries. Such a system decomposition has to balance site autonomy with the desirable but conflicting goal of system-wide uniformity in human and programming interfaces. The *cell* mechanism of AFS is an example of a mechanism that allows decentralized administration.

The economic impact of growth is closely related to how cost is reflected. The optimal system design for an organization where all system costs are borne centrally will be different from one for an organization where some costs are borne centrally and others are passed on to the end users. The latter model is more common, and favors designs in which the cost of incremental growth is almost entirely borne by the individuals benefiting from that growth. This in turn implies that incremental growth should minimally impact central resources.

2.1.3. *Efficient search*

As the volume of data in distributed repositories grows it becomes increasingly difficult to precisely identify the data item one wants. The distributed file systems of today offer little by way of help in this area. A nationwide distributed file system like AFS is invaluable when one knows at least the approximate pathname of the files one is interested in. It is much less convenient when one only has a vague specification, such as ‘‘a paper on RPC by someone at Stanford.’’

Relational databases offer associative access but are constrained in the kinds of queries that are possible and in the kinds of data that can be searched efficiently. Their indexing structure is typically static and is oriented toward textual or numerical searches. Presenting a speech fragment, a rough outline of a face, or the approximate 3-D shape of a molecule to a distributed database and asking it to search nationwide for matches is not something that can be done efficiently today.

It will be extremely valuable to build data repositories that can be extended by *value-added search services.* A good example from a traditional domain is the legal system, where numerous cases are tried and decided each day. The text of the trial proceedings and judgements are in the public domain. Yet there are a number of competing companies, successful for many years, whose sole service is the indexing of these cases. Because the indexing scheme is sophisticated, the time a lawyer spends on searching for applicable precedents to his case is substantially reduced. With large enough volumes of data, the index becomes as precious as the data itself!

The need for efficient search is not, of course, limited to the legal domain. Virtually every area of human activity is likely to benefit from distributed repositories that facilitate the annotation and indexing of data by parties other than creators and end users.

2.2. Technology Trends

2.2.1. *Portable computers*

In the past the computer industry has usually used improvements in hardware technology to provide better performance at constant cost, or to lower cost for constant performance. Over the last few years, however, vendors have turned their attention to factors such as physical size, compactness, weight and power consumption. Some of the laptop computers of today are as powerful as their substantially larger and heavier desktop ancestors of a few years ago. Some vendors now have product lines where advances in technology are translated into system-level miniaturization at constant cost and performance.

Although laptop computers are popular, their mode of usage is still quite primitive. Shared data is usually accessed by manually copying relevant files onto the laptop’s local disk or floppy, operating disconnected, and copying back modified files upon reconnection. Explicit file transfer via a modem is a less frequently used alternative. In contrast to these manual strategies, the Coda file system facilitates the use of shared data on portable computers by simplifying pre-caching of files, allowing autonomous operation while disconnected, and transparently reintegrating changes upon reconnection. In the future, Coda and other similar systems will substantially increase the use of shared data repositories from portable machines.

2.2.2. *Diverse types of data*

Numerical and textual I/O have traditionally been the dominant types of interaction between computers and humans. But over the last decade, the decreasing cost of hardware has made it economically feasible to use data types that are better suited to human cognition.

The impact is most evident in computer output. Even the cheapest Apple MacIntosh today can render graphics effectively, and any MacIntosh application can intermix text and graphics in its presentation.

The emergence of *de facto* standards such as X, Display Postscript, and Microsoft Windows indicates a similar, if less pervasive, trend in other operating system environments. More sophisticated use of graphics, with greater computing demands, is found in CAD/CAM systems.

Less common, but growing in popularity, is the input of graphical images. Today, scanners for the MacIntosh can convert any printed page into machine-readable data. Applications can merge scanned-in data with text and graphics.

Sound is another type of data that is making the transition from esoteric to commonplace. Digital signal processing hardware for speech and music synthesis is now available even on relatively inexpensive Unix workstations, such as those from Next, Inc. A recent issue of the Usenix journal *Computing Systems* included an audio compact disc supplement consisting exclusively of computer-generated music!

The topic of multi-media data is the focus of many research projects today. It is reasonable to expect that at least some of these projects will yield new paradigms of data input and output, as well as applications that combine them with existing types of data. Although textual and numerical data are likely to remain dominant, it is clear that other forms of data will supplement them in the future.

What does this diversification portend for distributed data repositories? Since all data is represented as a sequence of bits at the lowest level, it may appear that the distributed file systems of today are perfectly capable of storing any kind of data. But such a reductionist view ignores the fact that all successful repositories do, in fact, make explicit or implicit assumptions about the data they store. These include assumptions about *physical characterstics* such as the *size* of data items, as well as assumptions about *usage characteristics* such as *spatial* and *temporal locality, longevity,* and *patterns of sharing.* Future respositories will have to pay attention to the physical and usage characteristics of new types of data if they are to remain viable.

2.2.3. *Diverse network characteristics*

There is a considerable amount of excitement today about the rapid strides being made in the field of networking. Nationwide networks at 45 Mb/s are a reality, and work is under way to provide gigabit connectivity. Projects such as Autonet [8] at DEC SRC and Nectar [1] at CMU are pushing the current limits of performance in local area networking. Continuing progress in the areas of optic fiber materials and interface technology indicates that this trend will continue for a long time to come.

There is considerable debate in the distributed systems community about how best to use this new-found opportunity. Should communication be considered essentially free? Or should one take a Malthusian view, and design conservatively in the expectation that increased usage and more demanding applications will rapidly saturate high-speed networks?

A consideration that is completely ignored in these debates is the fact that the quests for performance and span of access are usually in conflict. For example, when accessing data from a hotel room via a modem, one only has 2400 b/s connectivity. While this is an improvement over the 300 b/s that was commonplace a decade ago, it is still a far cry from a gigabit network.

Cordless networking technologies are another source of diversity. Portable computers with built-in cellular telephone modems are available today. Work is under way both at IBM Yorktown Heights and at

Xerox PARC to use infrared communication in the range from 100 Kb/s to 10 Mb/s for indoor use of handheld portable machines. The Swedish Institute of Computer Science is exploring the use of packet radio for distributed file access.

These observations imply that the design of distributed data repositories should not be predicated on the universal availability of fast networks (for whatever value of "fast" is appropriate at the time of the design). A design that exploits fast networks when available, but remains usable via lower speed networks is likely to be more successful. The absolute values of "fast" and "slow" will change with time, but there is little doubt that diversity in networking speeds and capabilities will persist into the forseeable future.

3. Current State of Art

The majority of distributed data repositories today are either file systems based on the Unix model, or databases based on the relational model. One can also find distributed file systems based on other models such as MS-DOS, and non-relational databases such as IMS from IBM. However most advanced distributed systems work has been done in the context of Unix file systems and relational databases.

The work on distributed file systems, exemplified by AFS and NFS, has focused primarily on transparency, performance and scalability. Work has also been done in the areas of security and heterogeneity. AFS-4, currently under development, focuses on precise emulation of Unix semantics and the use of transactional technology for fast crash recovery. Echo, also under development, provides precise Unix semantics in conjunction with high availability via read-write replication at the server and disk levels. Coda also provides high availability, but achieves it by a combination of read-write replication at servers and deferred write-back caching at clients. The latter mechanism allows totally disconnected clients to function effectively, thus supporting the use of portable computers.

All successful distributed file systems exploit locality of reference by extensive caching at clients. They also assume low levels of concurrent write-sharing, and in the case of Coda, infrequent closely-spaced sequential write-sharing. None of these systems provide any mechanisms to assist in search.

Distributed databases such as Oracle, Ingres, and SQL/DS span few nodes in comparison with distributed file systems such as AFS. But they are large-scale systems in the sense that they store very large amounts of data at each node, often far exceeding the amount of data stored at a typical AFS server. These systems use a function shipping approach since they expect little locality of reference. Search is supported in the form of queries expressed in a query language (typically SQL). Queries are executed by servers and the results shipped to clients.

A considerable amount of theoretical work and some experimental work, has been done on query decomposition and optimization. The goal of this work is to improve efficiency when the data referenced by a query spans multiple servers. The fact that queries are constrained by the syntax and semantics of the query language considerably simplifies such decomposition. Many important applications that use databases exhibit high degrees of concurrent write-sharing and a need for high availability. Considerable work has therefore been done in the areas of concurrency control, fault tolerance and failure recovery. Commercial systems such as Tandem and IBM System/88 [2] are able to provide very high availability through a combination of hardware and software techniques. Relational databases have historically been constrained in the kinds of data they can store effectively. Projects such as Postgres and Starburst aim to overcome this limitation by providing mechanisms for extensibility.

Over the last five years there has been growing interest in a class of repositories referred to as *object oriented repositories.* Unfortunately, the term has been used to refer to so many variant concepts that it is difficult to define precisely. The underlying theme of this work is the merging of programming language concepts such as inheritance and abstract data types with system-level concepts such as atomicity and permanence. This is still an immature area, with minimal attention being paid to issues of large scale distribution.

There has also been recent interest in the AI community in *knowledge bases*. The key idea here is that the storage repository embodies considerable semantic knowledge about the data it stores. Consequently it is able to enforce semantic integrity at a high level of abstraction using common sense knowledge as well as domain specific knowledge. Again, this is relatively young area, and with little attention being paid to issues of distribution.

4. Research Agenda

No distributed data repository in existence today adequately addresses the trends discussed in Section 2. These trends induce a set of basic research problems that must be solved in order to meet future demands on distributed data repositories. In addressing these problems, it is important to preserve the advances that have been made in the past decade in areas such as transparency, scalability, and performance.

4.1. *Adaptive system behavior*

Future distributed systems will have to exhibit adaptive behavior at many levels of abstraction. Consider, for example, the problem of accessing data across low-bandwidth networks. One way to address the problem is to use compression. Different compression algorithms or parameters may be optimal for different types of data. A viable strategy would be for higher levels of the system to recognize the type of data they are dealing with, pass this information to the lower levels, and for the latter to adjust their transmission strategy accordingly. This strategy is an instance of separation of policy and mechanism, with the higher levels providing policy and the lower levels providing the mechanism.

In the above scenario the higher levels of the system are unaware of the networking environment. The adaptation is occurring at the lower levels, based on information provided by the higher levels. A radically different approach would be for the higher levels of the system to be cognizant of the state of the world at the lower levels and to adapt accordingly. For example, at low bandwidth it may be more efficient to adopt a function shipping strategy rather than a data shipping strategy. This approach makes sense if there are easily recognizable boundaries in the computation at the client within which large amounts of data are accessed, but across which relatively small amounts of data are reused. This combination of circumstances occurs most frequently in search, and is the reason distributed databases adopt a function shipping strategy. In this scenario, lower levels of the system provide information to the higher level about the state of the network but the adaptation occurs entirely at the high level.

Making the choice between function and data shipping dynamic complicates strategies to make the system scalable. Much of the scalability of systems such as AFS and Coda arises from the fact that clients bear most of the burden, with server resources being used only where essential. Function shipping muddies this simple picture, since server resources are now used for actual computation, not just for file system housekeeping. For the system as a whole to remain scalable, the decision to use function shipping must take into consideration the current load on the server. On a lightly loaded server, it may be perfectly reasonable to have a server perform an intense computation on behalf of a client. This is not a wise strategy when the server is heavily loaded, even if the client is connected via a low-bandwidth network.

High-level adaptation is also possible while remaining wholly within the data shipping paradigm. The client cache manager could adopt different caching strategies for different types of data. It could also dynamically detect usage patterns and modify its caching policies accordingly. Unlike hardware caches, where execution cycles and state for cache management are at a premium, file caches can afford more sophisticated and expensive strategies.

4.2. *Secure remote computation*

Function shipping opens a Pandora's box from the point of view of security. Distributed file systems such as AFS and Coda base their security on the fact that no user computation is ever performed on the servers. As long as physical integrity of the servers is guaranteed, these systems are able to bound the damage that a malicious user can cause. Even if he is able to subvert the hardware and software on a workstation, the worst that can happen is the compromise of data writable or readable by the users of that workstation. The rest of the system remains secure. Once client-generated computations are allowed on servers, how is one to prevent a malicious client from exploiting a flaw in the server software and injecting a virus?

Distributed databases have (probably unwittingly!) addressed this problem by severely constraining what gets executed on a server. All requests are expressed in a query language that the system interprets. The limitations of the query language, and the fact that the system has full semantic knowledge of it, together allow the system to avoid the pitfall mentioned in the previous paragraph.

Unfortunately the need to support distributed search on arbitrary data types suggests that neither a query language approach nor a more general interpretive approach is likely to be flexible enough. Rather, a client should be able to compose arbitrary search code at runtime and have a server execute it. Even ignoring the obvious questions pertaining to the mechanics of compilation and execution, there remains the deeper problem of guarding against malicious side effects of compiled code.

One strategy that appears plausible is for the server to disable all system calls made by the untrusted search code. Each data item being examined is successively mapped into the address space of the process running that code, using an iterator mechanism. At the end of each iteration, the search code returns an indication of whether the current item matches the search. Malicious search code could, at worst, do two things. It could clobber the address space of its own process, and it could loop infinitely. The former is a risk even with non-malicious buggy code and is easily handled by the usual operating system firewall between process address spaces. The latter is an instance of unauthorized denial of resources. One approach to handling this problem would be to bound the time allowed for each iteration of the search, and to abort the search if this limit is exceeded.

The strategy described in the previous paragraph focused on function shipping of search code. Could one be more general? When one says "make vmunix" in a directory on one’s laptop in a hotel room, it would be impressive if the system could recognize that function shipping the *make* would be far more efficient than data shipping all the files missing from the local cache over a phone line. When connected to a high speed network, the same command issued from the same laptop should result in files being cached and the *make* being performed locally. Unlike search, arbitrary computations may need to modify permanent data on the servers. Disabling all system calls is therefore not a viable option. It is an open question whether there are subsets of system calls that are large enough to be useful yet provably secure when used by remote computations.

4.3. *Extensibility*

The idea of factoring out common functionality into a substrate and allowing customization of other functionality is an intuitively appealing one. This concept, along with the concept of inheritance, is at the core of the object oriented programming paradigm that has received wide attention in the last few years. The focus of activity in this area has hitherto been at the *language* level. However the concept is also applicable to the *runtime* level. A common example of this is the partitioning of modern operating systems into a machine-independent section and machine-specific *device drivers.* The latter encapsulate all knowledge of the devices they control, and interact with the machine-independent section via a standardized interface.

For runtime extensibility to be viable, certain conditions must be satisfied:

- First, the common part of the system must provide functionality that is really usable by a large number of customizations. Otherwise individual customizations will end up reimplementing this functionality themselves.
- Second, the standardized interface must allow the right kind of information to be communicated across it. An interface that is deficient in this respect will limit the kinds of customizations that can be done.
- Third, the decomposition should not degrade performance significantly. If the cost of crossing the interface is high, the design of the interface should be such that the number of such crossings is low. If crossing is inexpensive, the interface can be crossed more frequently.
- Finally, the human effort needed to exploit extension should be commensurate with the gains from it. A case in point is the *shell* in Unix. In theory, the shell is merely a user program and each user can write his own. In practice, however, hardly a handful of shells have been written over the history of Unix.

We believe that it is both advisable and feasible to build distributed data repositories that are extensible. The need to support diverse types of data with very different physical and usage characteristics argues for customized caching strategies, transmission protocols, and server access strategies. At the same time, at least three aspects of repositories can be type independent:

- The first is *data location.* Virtually all distributed file systems and databases exploit the fact that location information can be treated as a hint, since it is self-validating upon use. Location information can therefore be cached at clients and used without explicit mechanisms for cache coherence.
- The second is *authentication.* A user’s identity is the same regardless of the kind of data he is accessing. Note that this is not true of *protection,* since the set of operations on a data item

are type-specific.

- The third type-independent aspect is *system administration.* From the point of view of an operator who is performing backup, or a system administrator who is deciding how to allocate disk storage quotas, it is irrelevant whether the data items in question are Unix files, database tuples, or Landsat images.

Distributed file systems and databases have been successful in large part because they insulate users from considerations pertaining to distribution. Although this insulation is occasionally imperfect, the majority of users are able to write applications ignoring distribution. The implementors of file systems and databases on the other hand are willing to put in the effort to attain transparency with good performance because they are confident of the market for their product.

How can we realize a similar state of affairs for other kinds of data? The approach has to be *evolutionary.* It should be relatively simple to take a class of applications that manage a new kind of data locally and allow the data to be distributed using existing runtime support. This initial realization may not be particularly efficient. However, it should be possible to incrementally improve the runtime support for the new data type, using usage information maintained by the system. Such usage information may, for example, suggest a different caching strategy, or a different concurrency control technique. After some number of such refinements, runtime support for the new data type will be closely matched to its use. The goal one would like to strive toward is that the customization effort be incremental and the cost be reasonable relative to the gains.

5. Conclusion

To explore the issues described in this paper, we are initiating a new research project, *Odyssey*, at Carnegie Mellon University. Although the project is still in its infancy, the broad outline of the Odyssey architecture is emerging. It consists of a substrate that provides common functionality for all types of data, and an extension mechanism that allows exploitation of type-specific properties. This decomposition into substrate and extension is present in all the major components of the system: the server, the client, and the network transport. A secure remote-execution mechanism allows servers to execute limited kinds of client-generated computation without fear of contamination. Data on servers is organized into typed volumes that can be manipulated by the administrative operations of the substrate, as well as type-specific operations invoked by clients.

An important goal of Odyssey is to simplify the creation and use of distributed repositories whose characteristics are well-tuned to the data stored in them. Today, the implementation of such a repository is done in isolation, with virtually no sharing of code or uniformity of administrative functions with other repositories. Consequently, the creation of a new kind of repository is a major undertaking, with little opportunity for incremental evolution. The Odyssey architecture provides a unifying framework, but allows diversity in areas where unification would be counter to performance or functionality.

In summary, this paper has argued that the problems of adaptive system behavior, secure remote computation, and extensibility are central to the future evolution of large-scale distributed data repositories. Although these problems have been identified, the best solutions to them are far from being clear. Only implementation and usage experience from the next generation of research prototypes will provide that information.

References

[1] Arnould, E.A., Bitz, F.J., Cooper, E.C., Kung, H.T., Sansom, R.D., Steenkiste, P.A.
The Design of Nectar: a Network Backplane for Heterogeneous Multicomputers.
In *Proceedings of the Third International Conference on Architectural Support for Programming Languages and Operating Systems, Boston, MA.* 1989.

[2] Harrison, E.S., Schmitt, E.J.
The Structure of System/88, a Fault-Tolerant Computer.
IBM Systems Journal 26(3), 1987.

[3] Hisgen, A., Birrell, A., Mann, T., Schroeder, M., Swart, G.
Availability and Consistency Tradeoffs in the Echo Distributed File System.
In *Proceedings of the Second Workshop on Workstation Operating Systems, Pacific Grove, CA.* September, 1989.

[4] Lindsay, B.
A Data Management Extension Architecture.
In *Proceedings of the 1987 ACM SIGMOD Conference on Management of Data*. 1987.

[5] Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., Lyon, B.
Design and Implementation of the Sun Network Filesystem.
In *Summer Usenix Conference Proceedings, Portland*. 1985.

[6] Satyanarayanan, M., Kistler, J.J., Kumar, P., Okasaki, M.E., Siegel, E.H., Steere, D.C.
Coda: A Highly Available File System for a Distributed Workstation Environment.
IEEE Transactions on Computers 39(4), April, 1990.

[7] Satyanarayanan, M.
Scalable, Secure, and Highly Available Distributed File Access.
IEEE Computer 23(5), May, 1990.

[8] Schroeder, M.D., Birrell, A.D., Burrows, M., Murray, H., Needham, R.M., Rodeheffer, T.L., Satterthwaite, E.H., Thacker, C.P.
Autonet: a High-speed, Self-configuring Local Area Network Using Point-to-point Links.
Technical Report 59, Digital Equipment Corporation, Systems Research Center, April, 1990.

[9] Stonebraker, M.
The Design and Implementation of Distributed INGRES.
The Ingres Papers: Anatomy of a Relational Database System.
In Stonebraker, M.,
Addison Wesley, 1986, Chapter 9.

[10] Stonebraker, M., Rowe,L.
The Design of POSTGRES.
In *Proceedings of the 1986 ACM SIGMOD Conference on Management of Data*. 1986.

Table of Contents

1. Introduction **1**

2. Future Demands **1**

2.1. Usage Trends **1**

2.1.1. ***Accessing data from anywhere*** **1**

2.1.2. ***Increasing scale*** **2**

2.1.3. ***Efficient search*** **2**

2.2. Technology Trends **3**

2.2.1. ***Portable computers*** **3**

2.2.2. ***Diverse types of data*** **3**

2.2.3. ***Diverse network characteristics*** **4**

3. Current State of Art **5**

4. Research Agenda **6**

4.1. ***Adaptive system behavior*** **6**

4.2. ***Secure remote computation*** **7**

4.3. ***Extensibility*** **8**

5. Conclusion **9**

References **10**