USENIX
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

The following paper was originally published in the

Proceedings of the Embedded Systems Workshop

Cambridge, Massachusetts, USA, March 29–31, 1999

Massively Distributed Systems: Design Issues and Challenges

Dan Nessett
3Com Corporation

For more information about the USENIX Association:
Phone: 1 510 528 8649 FAX: 1 510 548 5738
Email: office@usenix.org WWW: http://www.usenix.org

Massively Distributed Systems: Design Issues and Challenges

Dan Nessett
Technology Development Center
3Com Corporation
Dan_Nessett@3com.com

Abstract

The forty year trend in the computing industry is away from centralized, high unit cost, low unit volume products toward distributed, low unit cost, high unit volume products. The next step in this process is the emergence of massively distributed systems. These systems will penetrate even more deeply into the fabric of society and become the information power grids of the 21st century. They will be ubiquitous. Most will operate outside the normal cognizance of the people they serve and most will be based on embedded systems that present non-traditional computing interfaces to their users. They will be engineered to operate as distributed utilities, much like the energy, water, transportation and media broadcast businesses do today. The first deployment of massively distributed systems is likely to occur as support structures for these industries.

Massively distributed systems will differ from existing distributed systems in important ways. Such systems eventually will interconnect billions of nodes. This will necessitate changes in the way nodes interact with one another. One-to-many communications will be the norm, rather than one-to-one. The size of on-line application communities will necessitate the use of statistically correct rather than deterministic algorithms for resource accounting, fault detection and correction, and system management. These communities will coalesce and dissolve rapidly in order to host events that are of interest to groups formed specifically for this purpose. This will require new approaches to naming, routing, security and privacy, resource management and synchronization. Heterogeneity will be even more of a factor in the design, implementation and operation of massively distributed systems

This paper explores the nature and characteristics of massively distributed systems by proposing some examples and then using them to characterize nine major design issues. Seven of these are drawn from seminal work in the area of distributed systems. Two others are based on experience in distributed system design and implementation subsequent to that work.

1. Introduction

This paper explores trends in distributed system architecture and implementation that will have far-reaching consequences in the next 5-10 years. The paper's central thesis is that applications will become massively distributed during this interval and in so doing generate significant engineering problems, the solution of which will change the nature of distributed computing. Following this trend, networking technology will change in order better to serve the communications requirements of this new class of applications.

One major driver of massively distributed applications is the advent of ubiquitous embedded systems. While embedded systems will not always communicate either with their peers or with traditional computing equipment, important applications will require the interconnection of them in large numbers. Several examples are given in section 2. Furthermore, these applications will require computing architectures that off-load heavy computational tasks from embedded systems onto more powerful and capable equipment. Consequently, massively distributed applications will present problems to architects and implementers that transcend those of stand-alone embedded system applications.

The paper examines the motivation for and engineering problems of massively distributed systems. In the next section, a case is made why massively distributed systems will arise. Section 3 presents an analysis of their engineering design issues. Finally, section 4 presents a summary of the paper.

2. Motivation

Massively distributed systems are the next step in the development of computing. From a commercial perspective, this process started with centralized, high unit cost, low unit volume systems in the fifties and sixties. It then moved to less centralized, but still more or less stand-alone systems in the seventies. In the eighties, Network Service Providers (NSPs), such as Compuserv, Genie and America On-line, became popular providing email and bulletin board services to businesses and some advanced home users. In the first half of this dec-

ade, internet access became popular, supporting truly distributed applications, such as the World Wide Web. In the first decade of the 21st century, massively distributed systems will appear and become ubiquitous. In direct contrast to the systems of the fifties through the early nineties, these systems will be decentralized, low unit cost and high unit volume. They will be pervasive. Most will operate outside the normal cognizance of the people they serve and most will be based on embedded systems that present non-traditional computing interfaces to their users. They will be engineered to operate as distributed utilities, much like the energy, water, transportation and media broadcast businesses do today. Indeed, the first deployment of massively distributed systems is likely to occur as support structures for these industries.

Massively distributed systems will not be limited in scope to the interconnection of and interaction between embedded systems alone. While large numbers of embedded systems will participate in massively distributed systems, at least some will interact with traditional computing equipment. The latter will provide control, information storage, intensive computation and other services to massively distributed applications. Embedded systems will provide specialized point-of-presence services.

Several hypothetical examples illustrate the future impact of massively distributed systems. These will arise as natural extensions to current practice. They will be used subsequently to illustrate some of the distinguishing characteristics of massively distributed systems, which are significantly different from those of smaller scale distributed systems operating today.

2.1. Ubiquitous supply-chain management

Managing large distribution processes in the face of increasing governmental regulation, resource scarcity and system complexity will become increasingly important and open up markets for massively distributed systems. The classic utility businesses, e.g., the electrical, water, natural gas, and telephone companies, will be joined by other flow based service industries, e.g., transportation conglomerates, oil companies (especially those which own and operate service stations), large retail companies, including food, clothing, soft and hard goods businesses, and the information industries, such as newspaper, television, entertainment and advertisement concerns to concentrate on improving their margins by the efficient control of their product flow. Supply-chain management will spread from manufacturing to the day-to-day operations of companies close to the consumer. This will require the development and deployment of massively distributed systems that reach into the home, office, public pathways, and private institutions.

While the problems presented by these massively distributed applications will display certain differences, there will be a surprising amount of commonality. Managing flows, whether they consist of information, paper, dishwashing detergent, gasoline, or other commodities, requires sensors at the points where the product is consumed, quickly configurable transportation systems to move those commodities (e.g., trucks, power switching systems, airplanes, a communications fabric), and control algorithms and equipment to match the ingress of raw materials to the egress of the consumed products.

In many distributed systems, such as those concerned with power, water, natural gas, vehicle fuel, and mass market goods delivery, embedded processors will measure resource use and relay consumption information up the supply-chain. Control systems will utilize this information to schedule raw material inventory replenishment and organize the management of resource flows. Embedded systems close to the consumer (e.g., in appliances, yard sprinkler systems, home heating systems) will schedule resource consumption to minimize resource demand spikes, thereby reducing the peak capacity requirements of delivery systems [1].

An important supply chain that will dramatically influence how massively distributed systems are implemented is information flow. The resource management systems mentioned in the previous paragraph are geographically hierarchical, since they control physical processes characterized by the movement of mass or energy. Information flows as easily over large as small distances, a characteristic that distinguishes it from these other supply chains. Consequently, information supply chains will present significantly different problems to designers and implementers of massively distributed systems. Embedded systems will play a large role in information supply chains, as discussed in the next section.

It will be necessary to use massively distributed systems to solve this new class of flow control problems, since the ratio of control points to production facilities will grow to be several orders of magnitude larger than it is today. This adjustment will encourage new control algorithms based not on deterministic concepts, but rather on statistical notions of correctness. Profits will be maximized by ensuring a less than 100% accounting of resource usage and payment, since reaching the last

few percent will impose significant marginal costs that will exceed the marginal return. Banks already employ this philosophy in response to ATM fraud. The money that could be used to make their equipment more secure earns more interest than the losses they would prevent. Of course, to ensure an equitable and legally defensible operation (not to mention keeping stockholders happy), control algorithms must eliminate systematic fraud whereby one or a small group of individuals consistently benefit from a system's non-deterministic behavior.

2.2. Agile distributed information services

Perhaps the most dramatic development of the past 5 years is the growth of information distribution channels based on the Internet and their rapid penetration into areas not normally associated with computing. The World Wide Web, email, and Internet news groups are prominent examples. This trend will not only continue it will accelerate. The maturation of multi-media broadcast technology running over the Internet, assisted by extremely high capacity networking infrastructure will encourage the formation of virtual information communities. Unlike more traditional groupings, these communities will be ephemeral, lasting anywhere from weeks or months to periods as short as several hours or less. Primitive examples of these communities now exist, e.g., everyone reading the same newspaper and all those watching the same TV program. However, the information communities of tomorrow will not be based on rigidly structured broadcast hierarchies; rather they will encourage interactions between people who are members of a community at a particular moment. Multicast interest groups centered around a wide variety of subjects will form, much as internet news groups form today. Small production companies will present a play, host a debate, cover a sporting event or sponsor an electronic interactive event, such as a networked game that millions play simultaneously. The large media conglomerates will gradually evolve into distributed information and entertainment production companies offering a wider variety of programming material than they do now.

Managing such rapidly changing information communities will require more capable networks than currently exist. Switching and routing equipment will include embedded systems to monitor resource use and adjust allocations to meet fluctuating demand. These systems will control resources that are traditionally static, such as device backplane bandwidth, encryption and compression hardware, and agile error detection/correction hardware. Embedded processors will run transportable code, such as Java, in order to support active networking [2].

To protect consumers from unwanted intrusions into their lives, products will be developed to filter the information flowing to an end-system. These filters will range from those that allow parents to control the information available to their children (eliminating offensive material such as pornography) to those that eliminate the probable increase in junk information broadcast to consumers. Specialized processors implementing pattern recognition algorithms and embedded in routing, switching and end systems will provide powerful and customizable filtering capabilities to achieve these objectives.

To solve the problem of locating communities that might interest an individual, services will form offering electronic real-time catalogues. These will be the successors of the WWW index and search engines that exist currently. Electronic advisory services will inform consumers which communities offer the best value, using a value metric tailored for the customer's specific objectives. Specialized processors embedded in database and information repository systems will continuously index and catalogue information as it arrives.

Needless to say, participation in these communities will require payment, ranging from that necessary simply to reimburse common carriers, when the content is free (e.g., broadcasts of school plays, candidate sponsored political events, religious events), to payments for exclusive information commodities (e.g., pay-per-view sporting events, specialized financial broadcasts). Such payments will require the development and deployment of an electronic monetary system that utilizes a combination of digital cash, electronic credit systems (i.e., electronic analogues of the credit card, bank line of credit, and checks), and point of presence payment terminals. Processors embedded in cash cards and other payment tokens will play an important role in such systems.

2.3. The new economics

Electronic commerce is now a common feature of many commercial enterprises. However, most designers and implementers of electronic commerce systems have concentrated on the provision of electronic payment. Other aspects of electronic commerce have received less attention, in particular, many components that are well matched to implementation on massively distributed systems. Commerce involves not only payment for goods and services, but also their creation, advertisement, delivery, maintenance, and disposal.

The downward spiraling cost of communication is changing profit models in many businesses. Hard product companies will change their businesses to adapt to the new micro-economic environment. The creation of hard goods will become increasingly automated. Their advertisement will become more interactive, consumers will use third party search and evaluation companies that recommend products according to a detailed specification. Since consumers are rarely expert enough to create a useful and detailed description of their requirements, such companies will develop interactive systems with friendly customer interfaces that will lead a customer in the development of these requirements.

Customers will not travel to a remote location to use these systems. They will be accessed through communications facilities terminated in the customer's home or place of business. Eventually, requirement specifications will drive the creation of the product at an automated facility, create a just-in-time delivery plan to move the product from the manufacturing facility to the shipping end-point, schedule the resources to conduct warranted and purchased maintenance on the product, and arrange for the disposal of old equipment the customer no longer needs (or that was traded in for new equipment). Companies will optimize costs by coordinating this with the warehousing and shipment of the purchased equipment as well as with the disposal of the equipment of other customers.

Embedded systems will play an important role in the new economics. Once a manufacturing facility receives a product specification, embedded processors in robotic equipment will tailor its mechanical assets to create that product. Embedded processors will sense resource usage and work with control systems to ensure parts are delivered when necessary. Embedded systems will monitor and schedule warehouse and delivery capacity, cooperating with control systems to move the product to the customer. Embedded systems in robots will manage the repair of products returned for maintenance.

3. Design issues

Massively distributed systems are not only quantitatively different, but also qualitatively different than the distributed systems in place today. Their size and scope introduce new problems in design and implementation. However, to a certain extent we can use the experience we have gained building moderately scaled distributed systems to guide us in solving these problems.

This section presents a taxonomy of design issues for massively distributed systems. It is based on seminal work [3] that analyzed the current generation of distributed systems. This work identified seven major design issues. Two additional issues are added to the seven previously identified based on experience subsequent to that work. The classical design issues are : 1) naming, 2) error control, 3) resource management, 4) protection (security and privacy), 5) synchronization, 6) objects (representation, encoding and translation), and 7) measurement, testing and debugging. The additional design issues are : 8) heterogeneity, and 9) scale.

Each of these issues is used to show how massively distributed systems differ significantly from existing distributed systems. For pedagogical reasons, the issues are addressed in the following order : scale, heterogeneity, objects (representation, encoding and translation), resource management, protection (security and privacy), naming, error control, synchronization, and measurement, testing and debugging.

3.1. Scale

The main characteristic of massively distributed systems is their scale. While current distributed system are composed of hundreds of nodes[1], during the first decade of the 21st century massively distributed systems will be composed of thousands to billions of nodes, supporting hundreds to millions of users [4].

The scaling issue for massively distributed systems has a number of dimensions. In the area of communications capacity, recent events in the telecommunications industry promise to dramatically change the available worldwide bandwidth for networking. For example, Project Oxygen of the CTR Group, Ltd. intends to lay fiber across the ocean floors resulting in bandwidth of 1.28 terabits/sec on any segment [5]. Current schedules call for the completion of an Atlantic ring by Q4 2000 and a Pacific ring by Q2 2001. The pricing schedules proposed by CTR Group will result in transoceanic bandwidth offered at 1.5% of current satellite circuit prices and 1% of marine cable prices.

Power, water, natural gas, vehicle fuel and mass market supply chains are naturally large scale. Traditional information supply chains (e.g., radio, television, motion picture, the printed news media) are broadcast based and characterized by a small producer-to-consumer ratio. However, this is undergoing radical change. The growth of the world wide web has greatly increased the pro-

[1] Some may object to this assertion, observing that the current Internet is composed of millions of nodes. However, the term distributed system here means a cohesive set of computing resources acting together to achieve a common objective. While the routing infrastructure of the Internet can be considered to be a massively distributed application, no other system of this scale exists currently.

ducer-to-consumer ratio in data communications and this trend will continue. There may even come a time when the ratio approaches 1. This one factor has the potential to completely change how information supply chains are implemented and could be the major driver behind massively distributed systems.

Geographically, massively distributed applications supporting information supply chains will be global in extent. A single application running over a massively distributed system will interconnect users of significantly different languages, cultures and views. Communication costs will be structured so that they scale logarithmically with the number of destinations, thereby utilizing efficiencies inherent in broadcast and multicast communications. Individual subscribers will be able affordably to communicate with one to a hundred other subscribers. One-to-many communications will be the norm, as opposed to one-to-one transfers, which are most common today.

3.2. Heterogeneity

While heterogeneity is an important design issue for existing distributed systems, its extent in massively distributed systems will be significantly greater. In particular:

- The communications infrastructure will be composed of channels of very different capacities. Very low bandwidth channels (e.g., 1200 BPS) will still be in use in developing nations during the next decade, while the developed world will support very high bandwidth channels in the core services (e.g., 1 Tbps). Communications between embedded systems and their peers or higher-level facilities will occur over channels of significantly lower bandwidth than that available in the core of the network. Building applications that run over bandwidth diverse communication infrastructure will require new ways of organizing data flows.

- Similarly, end-systems will possess a wide variety of presentation techniques, from interactive HDTV to personal digital assistants and personal phone equipment. Applications will combine these end-systems into coherent interactive subsystems. In fact, it is possible today to create a communications conferencing system that patches together video-teleconferencing equipment, multicast capable workstations and cellular phones.

- Participatory communities, formed on demand, will choose from an array of costing options for their members. Short-lived communities may elect a pay-per-use approach, which is suited to their ephemeral nature. Longer-lived communities may be more inclined to charge a subscription fee for their services. Some communities with external financial backing may choose to charge no fee, transferring value to their sponsors through advertising, indirect persuasion, or charitable benefit. Each of these costing models must be supported by the massively distributed system infrastructure in a way that allows such diverse activities to exist concurrently.

- To support cost recovery for distributed applications, several efforts are underway to build and deploy electronic monetary systems. Existing applications are able to use a single funds transfer system, since many serve a limited customer base. Massively distributed systems, on the other hand, will be global in scope and normally serve customers in many countries who use a large variety of payment systems. For some, only traditional methods, such as a credit card or cash payment card will be available; while others may use different electronic systems, such as digital cash, digital checks, or digital credit cards. Embedded systems will play an important role in the latter class of payment system. Massively distributed systems must accommodate the use of all such systems by a single application. This will necessitate the development of funds transfer transaction clearinghouses that will convert funds from one system to the other.

- Distributed systems currently run on equipment managed by different administrations, which desire to keep its use under their control. Current practice is to rely on the users of this equipment to properly employ it according to policies promulgated by these administrations. However, this approach is changing as organizational IT budgets become larger and as Board of Directors, stock holders and other interested parties hold management more strictly accountable for information processing equipment use. Organizationally imposed constraints on distributed applications have a deleterious effect on their operation. A clear example of this is how router based firewalls not only provide protection against intruder attacks, but also prevent certain distributed applications, such as those using UDP, Java applets or those using the X window system from running across firewall boundaries. Massively distributed systems will experience even greater impediments to their operation, since not only will locally imposed constraints interfere with their operation, but other constraints will hinder them, such as national restrictions on transborder data flows and encrypted

data as well as the mandated use of certain standards for communications protocols.

- Distributed applications are composed of software components[2] that run on various platforms and are organized into component classes (e.g., file servers, public key distribution systems, file system clients, various clients on embedded systems), each of which performs a different function. Current distributed applications either assume that all components run the same version of software, or are structured according to a simple client/server model that must concurrently accommodate different versions on a small number of system (e.g., two or three). Massively distributed applications, because of other heterogeneity requirements, such as the use of different presentation formats, costing and cost recovery schemes, naming techniques and administration constraints, will be constructed of a significantly larger number of component types. Components of these types in general will not run the same version of software. Designing and implementing these applications will therefore require engineering techniques that allow them to operate in the face of software versioning heterogeneity.

- One of the fundamental mistakes made when building distributed systems is ignoring legacy applications and infrastructure. This invariably leads to poor customer acceptance of new technology. Customers cannot simply discard their existing applications when new infrastructure becomes available. In many cases these applications and their supporting infrastructure represent billions of investment dollars and reimplementing them according to the interfaces presented by new technology is economically infeasible. Massively distributed systems will be no different in this regard. However, existing distributed systems generally must accommodate stand-alone legacy systems and applications. Massively distributed systems, on the other hand, must accommodate both stand-alone as well as current distributed system legacy infrastructure and applications.

- While most existing distributed applications run on a number of different computing platforms, they are generally limited to a small number of common families, e.g., Unix, Windows, and perhaps MVS. Massively distributed applications, on the other hand, will run not only on the legacy platforms, but also on a wide variety of embedded systems supported by their own proprietary operating systems and hardware (e.g., automobile control systems, PDAs). For example, a massively distributed application for remote conferencing may have components that run on workstations, on equipment provided by a manufacturer for the TV cable industry, on cellular phones, on PCS based personal communication devices, and so forth. This will increase the number of different implementations of the software for a single component type and thereby increase the effort necessary to ensure the application works correctly.

[2] The term “component” is used here in its general sense. Distributed system components may or may not be constructed as object-oriented software components.

3.3. Objects (representation, encoding and translation)

A variety of efforts are underway to determine the best programming model for distributed objects (e.g., CORBA, Java). Lessons from these investigations will directly affect how objects are handled in massively distributed systems. However, there are certain issues arising from the scale of these systems that will introduce new constraints on how objects are represented, encoded and translated.

- Massively distributed system by their nature will require the creation, movement, modification and destruction of massive objects, which conceivably could contain thousands to millions of other objects. The size of these objects will affect how they are represented. For example, it will be infeasible to incorporate a copy of each contained object within the massive object. Instead, object references will be used within massive objects to represent its constituents. These objects will themselves be constructed using references to their contained objects, leading to a hierarchical object anatomy. However, several well-known problems associated with referencing distributed objects will require attention. For example, implementers of massively distributed systems will have to solve the lost object problem, which occurs when an object is destroyed without destroying all its references in other objects. Relying on manual techniques to recover from this error condition will not be an option, because of its scale. Similarly, revoking access to an object will be much more problematic for massively distributed systems than for existing distributed systems. It will be infeasible to locate all references to an object and delete them, even if the underlying object representation allows that. While using access lists can mitigate the problem of revocation, their use in massively distributed systems will be problematic,

since their size may make this approach unattractive. This problem is discussed in more detail in section 3.5.

- Not only will the representation of massively distributed objects require new techniques, their presentation to users will also require innovation. Some researchers have examined this problem. A new class of user interface represents objects as virtual spaces. This technique is eminently suitable for presenting massively distributed objects to end users. For example, a first level object might be represented as a virtual world, its constituent objects as countries, then cities, streets, houses, rooms, and so on, the exact structure depending on the size of the first level object and its relationship to its constituents as well as the relationship of the constituents to each other. Such presentation paradigms will be required to make massively distributed objects accessible to the technologically naive user.

- Since massively distributed applications will be global in scope, their users will belong to multiple cultures and countries. This will increase the necessity for the internationalization of data. While current applications can be configured to use different character sets, depending on choices made when the system is installed, massively distributed systems will have to internationalize data on-the-fly. This is especially true of embedded systems, which commonly will exist in mobile equipment. There will be no point when a system administrator can select a particular internationalized presentation set. Users of massively distributed applications will come and go during its lifetime, requiring run-time configuration of this data. Perhaps more importantly, there will not be a single system administrator who controls a massively distributed application. Its control will be distributed as well. Not only will character set data require internationalization, the application will have to internationalize other data such as financial, length/mass/time, and timezone. For example, an application that presents financial data may have to convert between dollars, yen and marks, depending on where the end-point system is located and how it has been configured by the customer.

- The storage of massively distributed objects will require new techniques. Current object storage systems assume objects are located on resources controlled by a single administration. The storage of a single massively distributed object, however, may use the resources of many storage systems, controlled by different administrations. This will lead to new problems in object store performance, error control and correctness. Object implementers will have to deal with the internal synchronization of object constituents, in regards to their creation, movement, modification and destruction. They also will have to deal with these issues when designing and implementing object caches.

- The algorithms used to manipulate massively distributed objects will utilize techniques that differ from those commonly used today. They will be replaced by those that use multi-cast communications in order to avoid object components understanding the object's global architecture. Voting and distributed agreement algorithms will become common. Programmers will create active objects, i.e., those which include active on-going computing, that blur the distinction between data and processing. While some existing distributed systems support active objects, massively distributed systems will organize large parallel computations around this concept, forming these computations by creating a first level object containing a large number of active objects residing on geographically disperse and administratively heterogeneous systems.

3.4. Resource management

In order to design and build massively distributed applications, engineers will have to grapple with new problems in resource management. Many existing distributed systems operate according to a local resource control model. Local processing manages its own resources, interacting with other threads of control through message passing or remote procedure call/method invocation. In massively distributed systems, objects will be composed of resources located in a large number of different places. Controlling the resources associated with an object will only be possible through a distributed global object resource management mechanism. This will introduce several new issues in distributed system resource control:

- Routing will become an issue not only at the network layer of the distributed system, but also at the application layer. Composite objects containing active objects will require routing services so the composites can interact with each other, especially if objects are allowed to move. For example, an embedded system such as a mobile PDA will move and connect to different portals in a network. If upon connecting, objects are moved from the PDA to execution environments hosted near the portal, invocation of those object's methods by other objects

will require a routing protocol to identify a path between them. The interactions between active objects and passive objects will require routing so active objects can locate and contact passive object methods. Congestion control within composite objects will be an issue. Programmers and object implementers will organize objects to avoid computational hot spots, similar to those experienced by massively parallel algorithms. Due to their scale, object implementers will require adaptive routing and congestion control within massively distributed objects, manual configuration will not be an option.

- Resource management in a massively distributed system will interact strongly with its heterogeneous nature. Applications will compose resources under the control of many administrations into a single distributed resource. Management of the distributed resource must accommodate the usage policies of the separate administrations. For example, cost recovery for a massively distributed object must accrete and disseminate sufficient funds to cover the costs of the composite objects. Since massively distributed objects may be highly recursive, cost recovery will require the composition of recursively discovered composite costs. As objects evolve, their cost recovery anatomy will change, forcing object implementers to dynamically configure the object's cost recovery algorithm. Since it will be difficult and expensive to continually track the exact cost structure of an object at any point in time, cost recovery will use statistical algorithms, assessing charges and dispersing payments so that the owners of component resources receive their payments, but avoiding too fine an accounting of usage. Periodic audits will ensure systematic fraud is minimized. Operators of massively distributed objects will adopt a fixed cost model, allowing users of the object to flexibly utilize its resources without charging for its components use.

- Object implementers will devise algorithms enforcing restrictions on object use that are required by restrictions on their components. Unlike existing distributed resource management, these algorithms will have to be adaptive, since implementers will not know all possible component resource usage policies, *a priori*. Object implementers will be challenged to understand object behavior, since dynamically changing an object's components, which may introduce new resource usage restrictions, will make this difficult.

- Massively distributed systems will contain extremely large volumes of data and enormous processing power. Effectively managing these resources will challenge implementers. Distributed system architects will merge the two prevalent ways to organize computations in a distributed system, i.e., *move data to the processing* (used by NFS, World Wide Web, FTP, gopher) and *move processing to the data* (used by active networking and Java applets) into a single scheme. Such objects will be moved within the distributed system and carry with them both code and data [6]. If the object consists primarily of data, it will closely approximate moving data to the processing. If it consists primarily of code, it will closely approximate moving processing to the data. RPC and other procedure-oriented paradigms will be emulated by including a single high level instruction in the code section, specifically a procedure identifier, which will be interpreted at the RPC server.

- The deployment of massively distributed systems will encourage the trend towards a new valuation model for information. Specifically, the independent variable driving value will be how long the information has existed. Massively distributed systems will increase the difficulty in keeping information private. Consequently, customers will pay for fresh information, which will rapidly decrease in value once it is produced. For example, stock market ticker data is now available electronically, making automated trading programs possible. However, this implies that most of the value of the ticker data is consumed in a very short period of time. Its public release without cost routinely occurs today 15 minutes after its generation. As massively distributed systems become common, more and more stock traders will be forced to use automatic trading programs, which will reduce the value of the ticker data even faster. It is likely that eventually ticker data will be available without cost within a few minutes of its production. This paradigm will apply to other data, such as library searches, financial analyses and market research.

- Upgrading software in a massively distributed system will pose new problems for software companies. It will become more difficult to locate all users of a particular version of software and users will become less cognizant of the version they are using. This situation will arise because objects will contain and use other objects implemented by software not directly visible to the user. To meet this new challenge, software companies will move from a pure

product business model to a combination of product and service model. Indeed, this is already happening. Many software companies offers a subscription purchase agreement for their products that provides the customer with one year of upgrades. Engineers will design their software to periodically query maintenance centers that will upgrade it automatically, if the software's maintenance contract is still active. Again, this is current practice. For example, the virus detection software that is part of the Norton System Tools product for Windows 95/98 periodically communicates with a maintenance site on the Internet and downloads new virus checking features as they are released. Customers will be forced to use maintenance contracts to keep their applications running. In the future, customers will force software manufacturers to provide open interfaces to their software and negotiate maintenance agreements with secondary companies, much like the hardware business does today.

- Massively distributed systems will force a reexamination of the way communication interfaces work. In particular, they will have to scale to accommodate a large number of correspondents. For example, a simple mass-market purchase application that allows customers to choose between several products may receive millions of requests in a short period of time. Current programming interfaces to communication services are totally inadequate to handle such a high rate of transactions. Furthermore, individually acknowledging each of these transactions would impose a significant processing burden on the systems running the application. This will encourage engineers to design communication interfaces and protocols that are better suited to high rate and volume transactions.

- Since massively distributed systems will dramatically increase the amount of traffic handled by a communications fabric, engineers will investigate how to efficiently move extremely large volumes of traffic over the internet. In addition much of this traffic will belong to one of several different quality of service classes (e.g., best effort delivery data, stream data) and so understanding how to accommodate different quality of service parameters in the internet is a problem with high priority. Fortunately, this problem is currently receiving significant attention by the Internet Engineering Task Force and the IEEE.

- Massively distributed systems will support a volume of information flow to an end-system beyond that which a user is capable of manually examining for interest, usefulness, or suitability. Various filtering techniques will be developed and used by customers to ensure their systems only present to them information in which they are interested. Additionally, once these filtration systems are common, applications will use feedback controls to gather and employ end-system supplied filtration data to decrease the amount of information sent to an end-point that is automatically discarded. Massively distributed systems will provide tools to applications that allow them to support this type of customization for large numbers of end-point systems.

3.5. Protection (security and privacy)

Protection of distributed system assets, including base resources such as processing, storage, communications and user-interface I/O as well as higher-level composites of these resources, e.g., processes, files, messages, display windows and more complex objects is not a strength of existing distributed systems. While engineers are currently engaged in developing solutions to the many problems that exist in this area, they are not addressing protection issues that will arise as massively distributed systems become prominent. Specifically:

- Massively distributed systems for the most part will support a large number of end-systems, many of which will be embedded in other equipment and used by technologically naive customers. These systems will require management, which very probably will occur through the use of systems under the control of service providers. Since many end-systems will either generate or contain data that customers consider private, the management technology for massively distributed systems must guarantee, to a reasonable extent, that unaggregated data is not compromised either by individuals operating the management sub-system or by the service providers as part of their corporate strategy, e.g., during the collection of marketing data. To meet this objective, end-systems must be customer anonymous with respect to the management sub-system. That is, there must be no tie between the identifiers used to manage end-systems and those used for billing, warranty or other services that require the identification of the individuals who own or use these end-systems.

- Since one-to-many communications will play a major role in massively distributed systems, engineers will have to address the problems of deletion, modification, insertion, replay, release of secret state, and

masquerade for this type of communications. Existing techniques are designed to protect one-to-one communications against these threats. New protocols and processing algorithms will be required to provide these services for one-to-many communications.

- The scale of massively distributed systems will introduce new problems in resource access control for both end-systems and infrastructure support systems. Currently, most systems use access control lists or their approximations (e.g., Unix file access permission bits). However, massively distributed systems will support millions of principals, which would lead to access control lists of unmanageable size, if implemented as they are currently. Implementers will require new techniques, such as access list caching hierarchies, capability systems that solve the revocation and lost object problems, and access control techniques that combine the advantages and characteristics of access control lists with capability access control and eliminate the disadvantages of each. Since massively distributed systems will be highly heterogeneous and require the support of legacy distributed systems, engineers will be forced to grapple with the hazards of composing systems that use access control lists with those using capability based access control [7].

- When confronted with the problem of information protection, existing distributed systems already must cope with the controls governments have placed on cryptographic technology. These constraints have hampered engineers in providing appropriate levels of security to the users of distributed systems. Massively distributed systems will generate new problems in this area. Since communications will occur between large numbers of end-systems, security service implementations will be hard pressed to ascertain which systems are constrained by particular national laws concerning encryption and what those limitations might be. As mobile systems become more prevalent and as they are integrated into massively distributed systems, enforcement of these constraints, even if they are known, would require tracking the position of a mobile system, identifying the constraints imposed by its locality, and communicating these constraints to all systems interacting with it. For the number of end-systems that will engage in a massively distributed application and for the frequency with which a mobile system's position might change (e.g., those used from or embedded in an automobile, train, boat or airplane), this will be technically infeasible. Since enforcing these legal constraints will be just as hard as implementing the technology that satisfies them, over time the constraints may be relaxed or even eliminated. However, while they are in force, the technical problems they raise will be formidable.

- Commercial enterprises will play a large role in massively distributed applications. Consequently, new threats will arise as the value of these applications increases. One concern is the security of internetwork routing. Massively distributed systems will introduce new factors into this problem. The infrastructure required for such systems will necessarily federate other large networks into a global communications fabric. However, routing service providers may wish to limit the traffic that transits their networks and subscribers may demand guarantees that the quality of service advertised by routing service providers is actually delivered. The former problem has received a great deal of attention, leading to the formulation of policy routing techniques. The latter problem on the other hand, has received less attention and will require further study.

3.6. Naming

Engineers have devoted considerable attention to the issue of naming in distributed systems over the last 20 years. Pioneering efforts, such as Grapevine [8], the Domain Name System (DNS) [9] and NIS [10], developed the concepts for the next generation of distributed naming systems, such as X.500 and LDAP accessed directories. To some extent these second-generation systems, especially X.500, are designed for large scale distributed systems. In particular, X.500 supports the storage and retrieval of customer defined data structures, a hierarchically structured and distributed naming context mechanism and decentralized authentication services. The Open Group's Distributed Computing Environment (DCE) uses either X.500 or DNS to construct a large scale naming system from its local Cell Directory System [11].

Even though the designers of second generation distributed naming systems have attempted to accommodate massively distributed systems in their design, there are a number of naming issues that transcend these designs and require attention:

- The volume of data within a massively distributed naming system will be so large, tens to hundreds of millions of entries, that customers will not know where to start when looking for an item. Hierarchical structures have the advantage of logarithmically

scaling the name space, given a specific choice of how the data should be organized. They have the disadvantage of constraining efficient queries to those whose unknowns are at the bottom of the hierarchy. Since a single massively distributed system will be used for a wide variety of purposes, multiple indexes into its naming data will be required. Independent services will manage these indexes and each will need to update their meta-data as the foundation data changes. This already exists for indexes of webpage data. However, as anyone who has used internet search engines realizes, the volume of data returned for many queries is too large for efficient use. Higher-level indexing services will use the services of lower-level services, leading to a hierarchy of naming data accessible by customers. This web will contain data of varying degrees of freshness, which will require new methods and algorithms to present a consistent and coherent view to customers.

- As stated previously, a large number of the organizational structures within a massively distributed system will be ephemeral, e.g., information communities which will unite around a particular event and then dissolve. However, many who would be candidates to participate in such an event may not be aware, *a priori*, that it exists. The scale of a massively distributed system invalidates traditional advertising techniques, since the volume and rate of the advertising information arriving at an end-system using these approaches would overload it. Consequently, engineers will devise new producer/consumer models of advertising. For example, one possibility is a hierarchically structured reverse advertising technique, whereby the customer advertises desirable product attributes to first tier brokers, which accrete these into reverse advertisements to higher level brokers. From other leaves in the hierarchy, product producers advertise their products to other first-tier brokers, which likewise accrete them into advertisements to higher level brokers. Somewhere within the hierarchy a rendezvous occurs and direct advertisements are sent or otherwise made available to customers. A simple example of this strategy for advertising and using computing resources has already been prototyped [12]

- Moving objects requires moving names embedded in the object. This necessitates either retaining the context in which those names are interpreted or translating them into contexts available at the new object site. Massively distributed systems makes either strategy difficult. Retaining contexts will compel an object to maintain contact with them as it moves around, which will in turn require meta-naming services (i.e., to name the naming context). Translating naming contexts will necessitate the discovery of the relationships between the different contexts. Furthermore, object sharing requires the naming of objects by different entities. Communicating names for the purpose of sharing requires moving the associated naming contexts.

- Different cultures may require different naming schemes, some hierarchical, some local. These must be coalesced into a usable massively distributed system naming scheme. The same object may be named differently depending on the use of the name and its characteristics (e.g., native language, available character set).

3.7 Error Control

To a certain extent the problem of error control in global networks has not been solved for existing distributed systems. For example, multicast links fail during video teleconferences, ftp sites become unreachable during a transfer due to firewall, router or link failure, and end-system failure results in broadcast storms that cripple local communications. All of these events are difficult to analyze and correct automatically. They invariably require the use of out-of-band channels to notify a responsible technician or system administrator. In a massively distributed system, however, the identification and use of an appropriate out-of-band channel will be problematic. Furthermore, service providers must provide error control in ways that conform to other requirements, such as privacy protection, high performance and scalability. Such constraints lead to the following problems:

- Fault isolation and correction in massively distributed systems will require new infrastructure services to monitor communications quality and deliver exception alarms to service providers when quality falls below a given threshold. Since the number of distributed applications running in a massively distributed system will be too large for manual monitoring to be cost effective, service providers and distributed system implementers will design, implement and deploy automatic fault detection and isolation mechanisms to ensure service remains available during various resource outages. This will be especially important for distributed embedded systems. The functions of existing network operation centers will become automated. Network operation centers will focus on higher level fault isolation and control,

such as rerouting communication services when a catastrophic event takes down large subsystems.

- Engineers of massively distributed systems will introduce new ways to process a large volume of information and number of transactions. As previously described for costing strategies, statistical algorithms will be used to cope with the decreasing probability that all necessary fault identification information is available in a reasonable amount of time. For example, consider the problem of automating the collection of payments for a video service. Errors in transmission, bugs in software, unresponsiveness of end-system equipment due to malfunction and operator error will lead to a certain amount of imprecision in the accounting and collection processes. An automated billing and collection application will record all funds received, match them with outstanding balances on accounts and attempt to resolve the amount received with the overall outstanding balance. A discrepancy below a certain threshold will be charged to the cost of doing business. Auditing processes will attempt to ensure that these discrepancies are not systematically occurring as the result of fraud. Such techniques will also be used for automated opinion polling, physical resource flow monitoring (e.g., power, water), and distributed system infrastructure maintenance.

- Massively distributed systems will require the use of highly available subsystems, including a redundant communications fabric provided by multiple service providers, fault-tolerant distributed processing for control and accounting, and high availability end-systems for content providers, which will be the next logical step beyond the high availability file systems currently deployed. Fault-isolation will have to operate in the face of privacy services, such as encryption, that are virtually non-existent today, but that will become prominent in the next decade. Service providers will have to respond to subscriber problem reports, even when the ultimate source and destination of traffic causing the problem is unknown. They will gravitate toward a scaled cost structure for various grades of service, from high grade indemnified service ("you lose service, we lose revenue"), to lower grades of best effort service.

- Detection of inconsistent state will be difficult in massively distributed systems, since the distributed state will not be available for centralized processing. Consequently, algorithms for communications and processing will be developed that are tolerant of inconsistencies. They will use redundancy not only within communication protocols, but also within objects so their state driven algorithms are fault tolerant. To achieve this objective, there will be an increased use of voting protocols and algorithms.

- It will be impossible to keep track of all the identifiers or other processing state associated with an object. Massively distributed systems must be designed to work in the face of lost objects, which will be the rule rather than the exception. Loss of identifiers will also increase the incidence of orphaned objects, which will increasingly require the use of global object garbage collection.

- One-to-many communications on a large scale will encourage the development of forward error correction techniques, which will replace feedback error control in many cases. Feedback error control requires the retention of a large amount of state and a high level of processing when applied to one-to-many communications. Idempotent operations will become the paradigm of choice when one-to-many communications support remote procedure or object method calls.

- Error control will move more and more into the application layer as the error characteristics of lower layers become heterogeneous (especially for one-to-many communications) and as applications are widely distributed. [13]. To manage application error control, management systems will expand their concerns from the network layer and below to all layers of the protocol hierarchy. This will lead to the standardization of management protocols and interfaces.

3.8 Synchronization

One of the most difficult problems that engineers of massively distributed systems will encounter is synchronizing computations consisting of thousands to millions of components. Current methods of synchronization, such as semaphores, monitors, barriers, remote procedure call, object method invocation, and message passing, do not scale well. Generally, they are suitable either for synchronizing closely coupled computations (e.g., semaphores, monitors and barriers), or for unicast distributed applications (e.g., remote procedure calls, object method invocations and message passing). Engineers have not yet devised efficient synchronization techniques for group computations, when groups contain thousands to millions of active elements. Specific problems in the area of synchronization include:

- Caching will become pervasive to accommodate performance, error control and resource management objectives. Keeping all cached copies synchronized will require new cache coherency algorithms, since simple write-back or write-through schemes do not scale properly. This will lead to synchronization marker algorithms where an authoritative copy of information is broadcast periodically to resync cached state. Resolution of differences will occur as a result of this synchronization activity. However, there will be many locations where portions of a large database are authoritative and redundancy in the data will accommodate failures.

- The scale of massively distributed applications will encourage engineers to use asynchronous computing, mimicking the way live organisms function. This will be a revolution in the way computations are designed, implemented and operated. It will no longer be possible to assume or reason that the state of some distributed application equals a given value at any point in time. Instead, engineers will design distributed application algorithms so that an invariant is always true with high probability.

- The use of pre-agreement algorithms based on roughly synchronized clocks will emerge as a major synchronization technique for massively distributed applications [14]. The accuracy and drift of local clocks will become a major engineering problem. Separate synchronization systems (e.g., the NIST time standard broadcast over WWV) will be used to decouple clock synchronization from resource depletion problems and other errors of massively distributed applications. The use of geographically distant physical clocks on high-precision real-time applications will require the transmission of both clock and location in a synchronization schedule to accommodate relativistic effects.

- It will be virtually impossible for thousands of components to synchronize using communications when it is not possible to use an external synchronizing signal. If synchronized clocks are not achievable, for example, because some components do not have real-time clocks (e.g., simple embedded systems), barrier synchronization implemented as a hierarchical tree will be necessary. Because of the attendant performance degradation associated with barrier synchronization, it will only be employed at major synchronization points separated by long intervals of time.

- To accommodate the improbability of a component knowing all other components with which it should synchronize, multicast communications will become a major synchronization technique. Thus, a component might multicast a message that announces the occurrence of an event to a multicast group without knowing exactly which other components belong to that group. Use of this technique will rely on the designer making a trade-off between synchronization and resource consumption.

- Synchronization and fault tolerance will interact strongly. Synchronization mechanisms will be required to operate in the face of a certain level of component and communications failure. However, synchronization algorithms will be designed to move the distributed application towards a well-defined goal even when failures occur. Thus, massively distributed applications will make significant use of probabilistic algorithms that complete with a probability less than unity in a given length of time, but that ensure this probability always monotonically increases in time.

3.9 Measurement, testing and debugging

The implementation of massively distributed applications requires measurement, testing and debugging to ensure applications behave correctly and perform well. These services are also required to correct implementations when they fail to meet either of these criteria. The scale of massively distributed systems makes the execution of these tasks difficult. In particular:

- Gathering the necessary measurement data to determine whether massively distributed applications are performing properly will be problematic. Directing this data to a single destination will almost certainly interfere with the phenomena being measured. Consequently, measurement will occur through an auxiliary distributed application, running in tandem with the target application. The measurement application will itself be massively distributed, requiring its own control infrastructure as well as new monitoring techniques that allow it to maintain contact with the measured application even when parts of it are created, destroyed and moved.

- Testing massively distributed applications will require new techniques to ensure the application scales. Current approaches used by testing engineers, which concentrate on determining whether the application works properly on a single machine will be inade-

quate for massively distributed applications While there is current work to develop harnesses to test client/server based applications, the introduction of a third support process into the test harnessthat supports client/server interactions (e.g., a database containing public key certificates) is rare. Due to the expense of developing a dedicated test harness for massively distributed applications, engineers will be forced to test applications during large scale beta deployment. This is in fact becoming the norm for much mass market software.

- It may not be possible to debug a massively distributed application under full load, so engineers will be forced to make trade-offs between debugging and real-time fault isolation with correction. Traditional debugging techniques, such as breakpointing, do not scale. Therefore, trace-driven analytical techniques will predominate when debugging massively distributed applications. Since test harnesses of the requisite massive scale will be economically infeasible, engineers will be forced to use execution traces of live application runs. Such execution traces may produce a very large volume of data, so test engineers will have to develop agile filtering, coalescing and viewing tools.

4. Summary

The nature of this paper is exploratory and most assertions are presented without detailed justification. Technology forecasting is always risky and, in hindsight, rarely completely accurate. However, I hope the analysis at least stimulated the reader to think about massively distributed systems, how their characteristics will influence embedded system design and implementation, and how they may change the way we do computing in the future. Even if there is disagreement about its content, it has succeeded if it motivates readers to develop an alternative taxonomy of the design issues and challenges associated with massively distributed systems.

5. References

[1] Gustavsson, R., "Agents with power," Communications of the ACM, Vol 42, No. 3, March, 1999, pp. 41-47.

[2] Tennenhouse, D.L., Smith, J.M., Sincoskie, W.D., Wetherall, D.J., and Minden. G.L, "A survey of active network research," IEEE Communications Magazine, pages 80-86, January 1997.

[3] Watson, R. W., "Distributed system architecture model," in Distributed Systems, Architecture and Implementation, Springer-Verlag, Berlin, NY, 1981.

[4] Takada, H. and Sakamura, K., "Compact, low-cost, but real-time distributed computing for computer augmented environments," Proc. 5th IEEE Comp. Soc. Workshop on Future Trends of Dist. Comp. Sys., Cheju Island, Korea, Aug., 1995, pp. 56-63.

[5] Lange, L., "The Internet," IEEE Spectrum, January, 1999, pp. 35-40. (see also www.oxygen.org).

[6] Sadok, D. H., Kelner, J., and Silva, R.A., "A distributed programming platform using mobile agents," 3rd Inter. Symp. On Autonomous Decentralized Sys., April, 1997, pp. 103-110.

[7] Nessett, D.M., "Factors affecting distributed system security," IEEE Trans. Soft. Eng., vol. SE- 13, pp. 223-248, 1987.

[8] Birrell, A.D., Levin, R., Needham, R.M.., and Schroeder, M.D., "Grapevine: an exercise in distributed computing," Communications of the ACM, April, 1982, 260-274.

[9] Mockapetris, P.V., Dunlap, K.J., "Development of the domain name system," Proc. of SIGCOMM '88, ACM, August 16-19, 1988, pp.123-133.

[10] Weiss, P., “Yellow pages protocol specification,” Technical Report, Sun Microsystems, Inc., Mountain View, CA., 1985.

[11] Dilley, J., “Practical experiences with the OSF cell directory service,” Networked Systems Architecture, Hewlett-Packard, International Workshop OSF DCE, Karlsruhe, Germany, Oct., 1993

[12] Cappello, P., Christiansen, B., Neary, M., and Schauser, K., "Market-based massively parallel internet compuing," Proc. 3rd Working Conference on Massively Parallel Programming Models, London, England, Nov., 1997, IEEE Computer Society, pp. 118-129.

[13] Saltzer, J., Reed, D., and Clark, D., “End-to-end arguments in system design,” ACM Transactions on Computer Systems 2(4), Nov., 1984, pp. 277-288.

[14] Kopetz, H., "The time-triggered architecture," Proc. 1st IEEE International Symposium on Object-oriented Real-time Distributed Computing, Kyoto, Japan, April, 1998, pp. 22-29.