Distributed Systems

Principles and Paradigms

Second edition

Adjusted for digital publishing

Andrew S. Tanenbaum
Maarten van Steen

To Suzanne, Barbara, Marvin, and the memory of Bram and Sweetie π
– AST

To Mariëlle, Max, and Elke
– MvS

Published by Maarten van Steen
This book was previously published by: Pearson Education, Inc.

ISBN: 978-15-302817-5-6 (printed version)

ISBN: 978-90-815406-0-5 (digital version)

Edition: 2. Printing: 01 (March 2016)

Contents

Chapter 1

Introduction

Computer systems are undergoing a revolution. From 1945, when the modern computer era began, until about 1985, computers were large and expensive. Even minicomputers cost at least tens of thousands of dollars each. As a result, most organizations had only a handful of computers, and for lack of a way to connect them, these operated independently from one another.

Starting around the the mid-1980s, however, two advances in technology began to change that situation. The first was the development of powerful microprocessors. Initially, these were 8-bit machines, but soon 16-, 32-, and 64-bit CPUs became common. Many of these had the computing power of a mainframe (i.e., large) computer, but for a fraction of the price.

The amount of improvement that has occurred in computer technology in the past half century is truly staggering and totally unprecedented in other industries. From a machine that cost 10 million dollars and executed 1 instruction per second, we have come to machines that cost 1000 dollars and are able to execute 1 billion instructions per second, a price/performance gain of 10^13. If cars had improved at this rate in the same time period, a Rolls Royce would now cost 1 dollar and get a billion miles per gallon. (Unfortunately, it would probably also have a 200-page manual telling how to open the door.)

The second development was the invention of high-speed computer networks. **Local-area networks** or **LANs** allow hundreds of machines within a building to be connected in such a way that small amounts of information can be transferred between machines in a few microseconds or so. Larger amounts of data can be moved between machines at rates of 100 million to 10 billion bits/sec. **Wide-area networks** or **WANs** allow millions of machines all over the earth to be connected at speeds varying from 64 Kbps (kilobits per second) to gigabits per second.

The result of these technologies is that it is now not only feasible, but easy, to put together computing systems composed of large numbers of computers connected by a high-speed network. They are usually called computer networks or **distributed systems**, in contrast to the previous **centralized systems** (or (**single-processor systems**) consisting of a single computer, its peripherals, and perhaps some remote terminals.

1.1 Definition of a distributed system

Various definitions of distributed systems have been given in the literature, none of them satisfactory, and none of them in agreement with any of the others. For our purposes it is sufficient to give a loose characterization:

> *A distributed system is a collection of independent computers that appears to its users as a single coherent system.*

This definition has several important aspects. The first one is that a distributed system consists of components (i.e., computers) that are autonomous. A second aspect is that users (be they people or programs) think they are dealing with a single system. This means that one way or the other the autonomous components need to collaborate. How to establish this collaboration lies at the heart of developing distributed systems. Note that no assumptions are made concerning the type of computers. In principle, even within a single system, they could range from high-performance mainframe computers to small nodes in sensor networks. Likewise, no assumptions are made on the way that computers are interconnected. We will return to these aspects later in this chapter.

Instead of going further with definitions, it is perhaps more useful to concentrate on important characteristics of distributed systems. One important characteristic is that differences between the various computers and the ways in which they communicate are mostly hidden from users. The same holds for the internal organization of the distributed system. Another important characteristic is that users and applications can interact with a distributed system in a consistent and uniform way, regardless of where and when interaction takes place.

In principle, distributed systems should also be relatively easy to expand or scale. This characteristic is a direct consequence of having independent computers, but at the same time, hiding how these computers actually take part in the system as a whole. A distributed system will normally be continuously available, although perhaps some parts may be temporarily out of order. Users and applications should not notice that parts are being

replaced or fixed, or that new parts are added to serve more users or applications.

In order to support heterogeneous computers and networks while offering a single-system view, distributed systems are often organized by means of a layer of software–that is, logically placed between a higher-level layer consisting of users and applications, and a layer underneath consisting of operating systems and basic communication facilities, as shown in Figure 1.1 Accordingly, such a distributed system is sometimes called **middleware**.

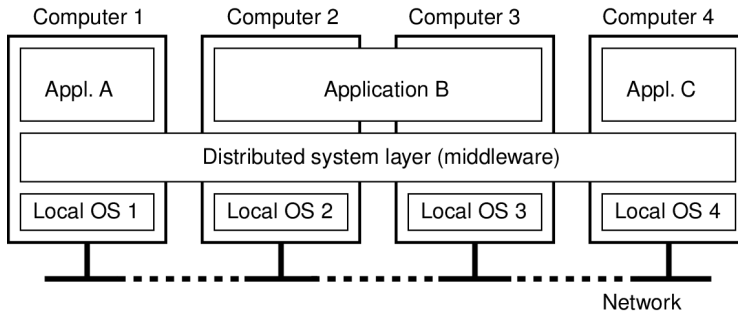

Figure 1.1: A distributed system organized as middleware. The middleware layer extends over multiple machines, and offers each application the same interface.

Figure 1.1 shows four networked computers and three applications, of which application *B* is distributed across computers 2 and 3. Each application is offered the same interface. The distributed system provides the means for components of a single distributed application to communicate with each other, but also to let different applications communicate. At the same time, it hides, as best and reasonable as possible, the differences in hardware and operating systems from each application.

1.2 Goals

Just because it is possible to build distributed systems does not necessarily mean that it is a good idea. After all, with current technology it is also possible to put four floppy disk drives on a personal computer. It is just that doing so would be pointless. In this section we discuss four important goals that should be met to make building a distributed system worth the effort. A distributed system should make resources easily accessible; it should reasonably hide the fact that resources are distributed across a network; it should be open; and it should be scalable.

1.2.1 Making resources accessible

The main goal of a distributed system is to make it easy for the users (and applications) to access remote resources, and to share them in a controlled and efficient way. Resources can be just about anything, but typical examples include things like printers, computers, storage facilities, data, files, Web pages, and networks, to name just a few. There are many reasons for wanting to share resources. One obvious reason is that of economics. For example, it is cheaper to let a printer be shared by several users in a small office than having to buy and maintain a separate printer for each user. Likewise, it makes economic sense to share costly resources such as supercomputers, high-performance storage systems, imagesetters, and other expensive peripherals.

Connecting users and resources also makes it easier to collaborate and exchange information, as is clearly illustrated by the success of the Internet with its simple protocols for exchanging files, mail, documents, audio, and video. The connectivity of the Internet is now leading to numerous virtual organizations in which geographically widely-dispersed groups of people work together by means of **groupware**, that is, software for collaborative editing, teleconferencing, and so on. Likewise, the Internet connectivity has enabled electronic commerce allowing us to buy and sell all kinds of goods without actually having to go to a store or even leave home.

However, as connectivity and sharing increase, security is becoming increasingly important. In current practice, systems provide little protection against eavesdropping or intrusion on communication. Passwords and other sensitive information are often sent as cleartext (i.e., unencrypted) through the network, or stored at servers that we can only hope are trustworthy. In this sense, there is much room for improvement. For example, it is currently possible to order goods by merely supplying a credit card number. Rarely is proof required that the customer owns the card. In the future, placing orders this way may be possible only if you can actually prove that you physically possess the card by inserting it into a card reader.

Another security problem is that of tracking communication to build up a preference profile of a specific user [Wang et al., 1998]. Such tracking explicitly violates privacy, especially if it is done without notifying the user. A related problem is that increased connectivity can also lead to unwanted communication, such as electronic junk mail, often called spam. In such cases, what we may need is to protect ourselves using special information filters that select incoming messages based on their content.

1.2.2 Distribution transparency

An important goal of a distributed system is to hide the fact that its processes and resources are physically distributed across multiple computers. A distributed system that is able to present itself to users and applications as if it were only a single computer system is said to be **transparent**. Let us first take a look at what kinds of transparency exist in distributed systems. After that we will address the more general question whether transparency is always required.

Types of transparency

The concept of transparency can be applied to several aspects of a distributed system, the most important ones shown in Figure 1.2.

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource is replicated
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource

Figure 1.2: Different forms of transparency in a distributed system [ISO, 1995].

Access transparency deals with hiding differences in data representation and the way that resources can be accessed by users. At a basic level, we wish to hide differences in machine architectures, but more important is that we reach agreement on how data is to be represented by different machines and operating systems. For example, a distributed system may have computer systems that run different operating systems, each having their own file-naming conventions. Differences in naming conventions, as well as how files can be manipulated, should all be hidden from users and applications.

An important group of transparency types has to do with the location of a resource. **Location transparency** refers to the fact that users cannot tell where a resource is physically located in the system. Naming plays an important role in achieving location transparency. In particular, location transparency can be achieved by assigning only logical names to resources,

that is, names in which the location of a resource is not secretly encoded. An example of a such a name is the URL *http://www.prenhall.com/index.html*, which gives no clue about the location of Prentice Hall's main Web server. The URL also gives no clue as to whether *index.html* has always been at its current location or was recently moved there. Distributed systems in which resources can be moved without affecting how those resources can be accessed are said to provide **migration transparency**. Even stronger is the situation in which resources can be relocated *while* they are being accessed without the user or application noticing anything. In such cases, the system is said to support **relocation transparency**. An example of relocation transparency is when mobile users can continue to use their wireless laptops while moving from place to place without ever being (temporarily) disconnected.

As we shall see, replication plays a very important role in distributed systems. For example, resources may be replicated to increase availability or to improve performance by placing a copy close to the place where it is accessed. **Replication transparency** deals with hiding the fact that several copies of a resource exist. To hide replication from users, it is necessary that all replicas have the same name. Consequently, a system that supports replication transparency should generally support location transparency as well, because it would otherwise be impossible to refer to replicas at different locations.

We already mentioned that an important goal of distributed systems is to allow sharing of resources. In many cases, sharing resources is done in a cooperative way, as in the case of communication. However, there are also many examples of competitive sharing of resources. For example, two independent users may each have stored their files on the same file server or may be accessing the same tables in a shared database. In such cases, it is important that each user does not notice that the other is making use of the same resource. This phenomenon is called **concurrency transparency**. An important issue is that concurrent access to a shared resource leaves that resource in a consistent state. Consistency can be achieved through locking mechanisms, by which users are, in turn, given exclusive access to the desired resource. A more refined mechanism is to make use of transactions, but as we shall see in later chapters, transactions are quite difficult to implement in distributed systems.

A popular alternative definition of a distributed system, due to Leslie Lamport, is "You know you have one when the crash of a computer you've never heard of stops you from getting any work done." This description puts the finger on another important issue of distributed systems design: dealing with failures. Making a distributed system **failure transparent** means that a

user does not notice that a resource (he has possibly never heard of) fails to work properly, and that the system subsequently recovers from that failure. Masking failures is one of the hardest issues in distributed systems and is even impossible when certain apparently realistic assumptions are made, as we will discuss in Chapter 8. The main difficulty in masking failures lies in the inability to distinguish between a dead resource and a painfully slow resource. For example, when contacting a busy Web server, a browser will eventually time out and report that the Web page is unavailable. At that point, the user cannot conclude that the server is really down.

Degree of transparency

Although distribution transparency is generally considered preferable for any distributed system, there are situations in which attempting to completely hide all distribution aspects from users is not a good idea. An example is requesting your electronic newspaper to appear in your mailbox before 7 AM local time, as usual, while you are currently at the other end of the world living in a different time zone. Your morning paper will not be the morning paper you are used to.

Likewise, a wide-area distributed system that connects a process in San Francisco to a process in Amsterdam cannot be expected to hide the fact that Mother Nature will not allow it to send a message from one process to the other in less than about 35 milliseconds. In practice it takes several hundreds of milliseconds using a computer network. Signal transmission is not only limited by the speed of light, but also by limited processing capacities of the intermediate switches.

There is also a trade-off between a high degree of transparency and the performance of a system. For example, many Internet applications repeatedly try to contact a server before finally giving up. Consequently, attempting to mask a transient server failure before trying another one may slow down the system as a whole. In such a case, it may have been better to give up earlier, or at least let the user cancel the attempts to make contact.

Another example is where we need to guarantee that several replicas, located on different continents, need to be consistent all the time. In other words, if one copy is changed, that change should be propagated to all copies before allowing any other operation. It is clear that a single update operation may now even take seconds to complete, something that cannot be hidden from users.

Finally, there are situations in which it is not at all obvious that hiding distribution is a good idea. As distributed systems are expanding to devices that people carry around, and where the very notion of location and context awareness is becoming increasingly important, it may be best to actually

expose distribution rather than trying to hide it. This distribution exposure will become more evident when we discuss embedded and ubiquitous distributed systems later in this chapter. As a simple example, consider an office worker who wants to print a file from her notebook computer. It is better to send the print job to a busy nearby printer, rather than to an idle one at corporate headquarters in a different country.

There are also other arguments against distribution transparency. Recognizing that full distribution transparency is simply impossible, we should ask ourselves whether it is even wise to *pretend* that we can achieve it. It may be much better to make distribution explicit so that the user and application developer are never tricked into believing that there is such a thing as transparency. The result will be that users will much better understand the (sometimes unexpected) behavior of a distributed system, and are thus much better prepared to deal with this behavior.

The conclusion is that aiming for distribution transparency may be a nice goal when designing and implementing distributed systems, but that it should be considered together with other issues such as performance and comprehensibility. The price for not being able to achieve full transparency may be surprisingly high.

1.2.3 Openness

Another important goal of distributed systems is openness. An **open distributed system** is a system that offers services according to standard rules that describe the syntax and semantics of those services. For example, in computer networks, standard rules govern the format, contents, and meaning of messages sent and received. Such rules are formalized in protocols. In distributed systems, services are generally specified through **interfaces**, which are often described in an **Interface Definition Language** (**IDL**). Interface definitions written in an IDL nearly always capture only the syntax of services. In other words, they specify precisely the names of the functions that are available together with types of the parameters, return values, possible exceptions that can be raised, and so on. The hard part is specifying precisely what those services do, that is, the semantics of interfaces. In practice, such specifications are always given in an informal way by means of natural language.

If properly specified, an interface definition allows an arbitrary process that needs a certain interface to talk to another process that provides that interface. It also allows two independent parties to build completely different implementations of those interfaces, leading to two separate distributed systems that operate in exactly the same way. Proper specifications are complete and neutral. Complete means that everything that is necessary

to make an implementation has indeed been specified. However, many interface definitions are not at all complete, so that it is necessary for a developer to add implementation-specific details. Just as important is the fact that specifications do not prescribe what an implementation should look like; they should be neutral. Completeness and neutrality are important for interoperability and portability [Blair and Stefani, 1998]. **Interoperability** characterizes the extent by which two implementations of systems or components from different manufacturers can co-exist and work together by merely relying on each other's services as specified by a common standard. **Portability** characterizes to what extent an application developed for a distributed system *A* can be executed, without modification, on a different distributed system *B* that implements the same interfaces as *A*.

Another important goal for an open distributed system is that it should be easy to configure the system out of different components (possibly from different developers). Also, it should be easy to add new components or replace existing ones without affecting those components that stay in place. In other words, an open distributed system should also be **extensible**. For example, in an extensible system, it should be relatively easy to add parts that run on a different operating system, or even to replace an entire file system. As many of us know from daily practice, attaining such flexibility is easier said than done.

Separating policy from mechanism

To achieve flexibility in open distributed systems, it is crucial that the system is organized as a collection of relatively small and easily replaceable or adaptable components. This implies that we should provide definitions not only for the highest-level interfaces, that is, those seen by users and applications, but also definitions for interfaces to internal parts of the system and describe how those parts interact. This approach is relatively new. Many older and even contemporary systems are constructed using a monolithic approach in which components are only logically separated but implemented as one, huge program. This approach makes it hard to replace or adapt a component without affecting the entire system. Monolithic systems thus tend to be closed instead of open.

The need for changing a distributed system is often caused by a component that does not provide the optimal policy for a specific user or application. As an example, consider caching in the World Wide Web. Browsers generally allow users to adapt their caching policy by specifying the size of the cache, and whether a cached document should always be checked for consistency, or perhaps only once per session. However, the user cannot influence other caching parameters, such as how long a document

may remain in the cache, or which document should be removed when the cache fills up. Also, it is impossible to make caching decisions based on the *content* of a document. For instance, a user may want to cache railroad timetables, knowing that these hardly change, but never information on current traffic conditions on the highways.

What we need is a separation between policy and mechanism. In the case of Web caching, for example, a browser should ideally provide facilities for only storing documents, and at the same time allow users to decide which documents are stored and for how long. In practice, this can be implemented by offering a rich set of parameters that the user can set (dynamically). Even better is that a user can implement his own policy in the form of a component that can be plugged into the browser. Of course, that component must have an interface that the browser can understand so that it can call procedures of that interface.

1.2.4 Scalability

Worldwide connectivity through the Internet is rapidly becoming as common as being able to send a postcard to anyone anywhere around the world. With this in mind, scalability is one of the most important design goals for developers of distributed systems.

Scalability of a system can be measured along at least three different dimensions [Neuman, 1994]. First, a system can be scalable with respect to its size, meaning that we can easily add more users and resources to the system. Second, a geographically scalable system is one in which the users and resources may lie far apart. Third, a system can be administratively scalable, meaning that it can still be easy to manage even if it spans many independent administrative organizations. Unfortunately, a system that is scalable in one or more of these dimensions often exhibits some loss of performance as the system scales up.

Scalability problems

When a system needs to scale, very different types of problems need to be solved. Let us first consider scaling with respect to size. If more users or resources need to be supported, we are often confronted with the limitations of centralized services, data, and algorithms (see Figure 1.3). For example, many services are centralized in the sense that they are implemented by means of only a single server running on a specific machine in the distributed system. The problem with this scheme is obvious: the server can become a bottleneck as the number of users and applications grows. Even if

we have virtually unlimited processing and storage capacity, communication with that server will eventually prohibit further growth.

Unfortunately, using only a single server is sometimes unavoidable. Imagine that we have a service for managing highly confidential information such as medical records, bank accounts, and so on. In such cases, it may be best to implement that service by means of a single server in a highly secured separate room, and protected from other parts of the distributed system through special network components. Copying the server to several locations to enhance performance may be out of the question as it would make the service less secure.

Concept	Example
Centralized services	A single server for all users
Centralized data	A single on-line telephone book
Centralized algorithms	Doing routing based on complete information

Figure 1.3: Examples of scalability limitations.

Just as bad as centralized services are centralized data. How should we keep track of the telephone numbers and addresses of 50 million people? Suppose that each data record could be fit into 50 characters. A single 2.5-gigabyte disk partition would provide enough storage. But here again, having a single database would undoubtedly saturate all the communication lines into and out of it. Likewise, imagine how the Internet would work if its Domain Name System (DNS) was still implemented as a single table. DNS maintains information on millions of computers worldwide and forms an essential service for locating Web servers. If each request to resolve a URL had to be forwarded to that one and only DNS server, it is clear that no one would be using the Web (which, by the way, would solve the problem).

Finally, centralized algorithms are also a bad idea. In a large distributed system, an enormous number of messages have to be routed over many lines. From a theoretical point of view, the optimal way to do this is collect complete information about the load on all machines and lines, and then run an algorithm to compute all the optimal routes. This information can then be spread around the system to improve the routing.

The trouble is that collecting and transporting all the input and output information would again be a bad idea because these messages would overload part of the network. In fact, any algorithm that operates by collecting information from all the sites, sends it to a single machine for processing, and then distributes the results should generally be avoided. Only decentralized algorithms should be used. These algorithms generally

have the following characteristics, which distinguish them from centralized algorithms:

1. No machine has complete information about the system state.
2. Machines make decisions based only on local information.
3. Failure of one machine does not ruin the algorithm.
4. There is no implicit assumption that a global clock exists.

The first three follow from what we have said so far. The last is perhaps less obvious but also important. Any algorithm that starts out with: "At precisely 12:00:00 all machines shall note the size of their output queue" will fail because it is impossible to get all the clocks exactly synchronized. Algorithms should take into account the lack of exact clock synchronization. The larger the system, the larger the uncertainty. On a single LAN, with considerable effort it may be possible to get all clocks synchronized down to a few microseconds, but doing this nationally or internationally is tricky.

Geographical scalability has its own problems. One of the main reasons why it is currently hard to scale existing distributed systems that were designed for local-area networks is that they are based on **synchronous communication**. In this form of communication, a party requesting service, generally referred to as a **client**, blocks until a reply is sent back. This approach generally works fine in LANs where communication between two machines is generally at worst a few hundred microseconds. However, in a wide-area system, we need to take into account that interprocess communication may be hundreds of milliseconds, three orders of magnitude slower. Building interactive applications using synchronous communication in wide-area systems requires a great deal of care (and not a little patience).

Another problem that hinders geographical scalability is that communication in wide-area networks is inherently unreliable, and virtually always point-to-point. In contrast, local-area networks generally provide highly reliable communication facilities based on broadcasting, making it much easier to develop distributed systems. For example, consider the problem of locating a service. In a local-area system, a process can simply broadcast a message to every machine, asking if it is running the service it needs. Only those machines that have that service respond, each providing its network address in the reply message. Such a location scheme is unthinkable in a wide-area system: just imagine what would happen if we tried to locate a service this way in the Internet. Instead, special location services need to be designed, which may need to scale worldwide and be capable of servicing a billion users. We return to such services in Chapter 5.

Geographical scalability is strongly related to the problems of centralized solutions that hinder size scalability. If we have a system with many cen-

tralized components, it is clear that geographical scalability will be limited due to the performance and reliability problems resulting from wide-area communication. In addition, centralized components now lead to a waste of network resources. Imagine that a single mail server is used for an entire country. This would mean that sending an e-mail to your neighbor would first have to go to the central mail server, which may be hundreds of miles away. Clearly, this is not the way to go.

Finally, a difficult, and in many cases open question is how to scale a distributed system across multiple, independent administrative domains. A major problem that needs to be solved is that of conflicting policies with respect to resource usage (and payment), management, and security.

For example, many components of a distributed system that reside within a single domain can often be trusted by users that operate within that same domain. In such cases, system administration may have tested and certified applications, and may have taken special measures to ensure that such components cannot be tampered with. In essence, the users trust their system administrators. However, this trust does not expand naturally across domain boundaries.

If a distributed system expands into another domain, two types of security measures need to be taken. First of all, the distributed system has to protect itself against malicious attacks from the new domain. For example, users from the new domain may have only read access to the file system in its original domain. Likewise, facilities such as expensive image setters or high-performance computers may not be made available to foreign users. Second, the new domain has to protect itself against malicious attacks from the distributed system. A typical example is that of downloading programs such as applets in Web browsers. Basically, the new domain does not know behavior what to expect from such foreign code, and may therefore decide to severely limit the access rights for such code. The problem, as we shall see in Chapter 9, is how to enforce those limitations.

Scaling techniques

Having discussed some of the scalability problems brings us to the question of how those problems can generally be solved. In most cases, scalability problems in distributed systems appear as performance problems caused by limited capacity of servers and network. There are now basically only three techniques for scaling: hiding communication latencies, distribution, and replication (see also Neuman [1994]).

Hiding communication latencies is important to achieve geographical scalability. The basic idea is simple: try to avoid waiting for responses to remote (and potentially distant) service requests as much as possible. For

example, when a service has been requested at a remote machine, an alternative to waiting for a reply from the server is to do other useful work at the requester's side. Essentially, what this means is constructing the requesting application in such a way that it uses only **asynchronous communication**. When a reply comes in, the application is interrupted and a special handler is called to complete the previously-issued request. Asynchronous communication can often be used in batch-processing systems and parallel applications, in which more or less independent tasks can be scheduled for execution while another task is waiting for communication to complete. Alternatively, a new thread of control can be started to perform the request. Although it blocks waiting for the reply, other threads in the process can continue.

However, there are many applications that cannot make effective use of asynchronous communication. For example, in interactive applications when a user sends a request he will generally have nothing better to do than to wait for the answer. In such cases, a much better solution is to reduce the overall communication, for example, by moving part of the computation that is normally done at the server to the client process requesting the service. A typical case where this approach works is accessing databases using forms. Filling in forms can be done by sending a separate message for each field, and waiting for an acknowledgment from the server, as shown in Figure 1.4. For example, the server may check for syntactic errors before accepting an entry. A much better solution is to ship the code for filling in the form, and possibly checking the entries, to the client, and have the client return a completed form, as shown in Figure 1.4. This approach of shipping code is now widely supported by the Web in the form of Java applets and Javascript.

Another important scaling technique is **distribution**. Distribution involves taking a component, splitting it into smaller parts, and subsequently spreading those parts across the system. An excellent example of distribution is the Internet Domain Name System (DNS). The DNS name space is hierarchically organized into a tree of **domains**, which are divided into nonoverlapping **zones**, as shown in Figure 1.5. The names in each zone are handled by a single name server. Without going into too many details, one can think of each path name being the name of a host in the Internet, and thus associated with a network address of that host. Basically, resolving a name means returning the network address of the associated host. Consider, for example, the name *nl.vu.cs.flits*. To resolve this name, it is first passed to the server of zone *Z1* (see Figure 1.5) which returns the address of the server for zone *Z2*, to which the rest of name, *vu.cs.flits*, can be handed. The server for *Z2* will return the address of the server for zone *Z3*, which is

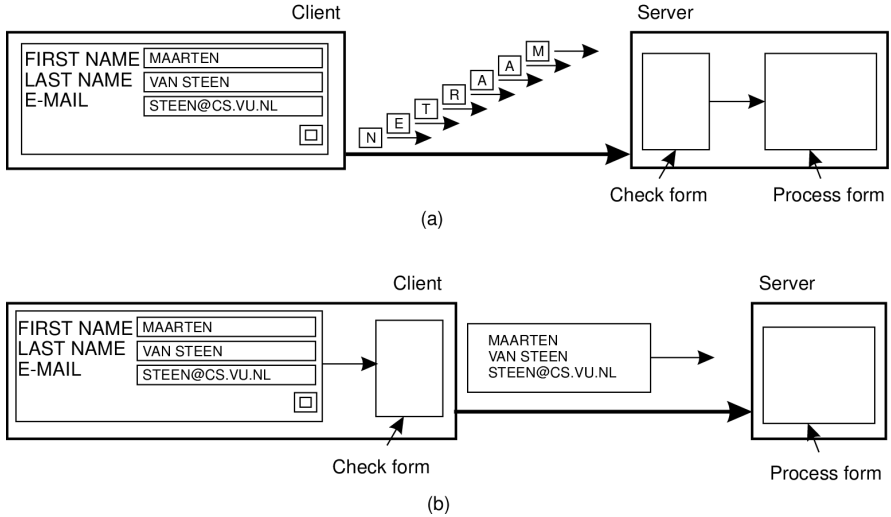

Figure 1.4: The difference between letting (a) a server or (b) a client check forms as they are being filled.

capable of handling the last part of the name and will return the address of the associated host.

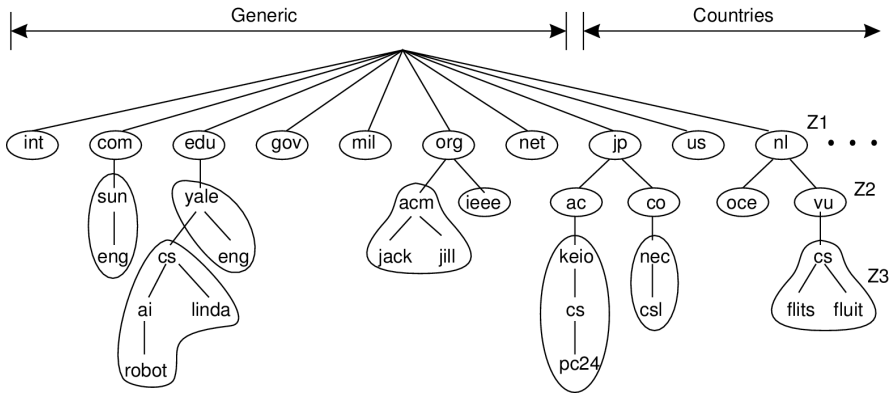

Figure 1.5: An example of dividing the DNS name space into zones.

This example illustrates how the *naming service*, as provided by DNS, is distributed across several machines, thus avoiding that a single server has to deal with all requests for name resolution.

As another example, consider the World Wide Web. To most users,

the Web appears to be an enormous document-based information system in which each document has its own unique name in the form of a URL. Conceptually, it may even appear as if there is only a single server. However, the Web is physically distributed across a large number of servers, each handling a number of Web documents. The name of the server handling a document is encoded into that document's URL. It is only because of this distribution of documents that the Web has been capable of scaling to its current size.

Considering that scalability problems often appear in the form of performance degradation, it is generally a good idea to actually **replicate** components across a distributed system. Replication not only increases availability, but also helps to balance the load between components leading to better performance. Also, in geographically widely-dispersed systems, having a copy nearby can hide much of the communication latency problems mentioned before.

Caching is a special form of replication, although the distinction between the two is often hard to make or even artificial. As in the case of replication, caching results in making a copy of a resource, generally in the proximity of the client accessing that resource. However, in contrast to replication, caching is a decision made by the client of a resource, and not by the owner of a resource. Also, caching happens on demand whereas replication is often planned in advance.

There is one serious drawback to caching and replication that may adversely affect scalability. Because we now have multiple copies of a resource, modifying one copy makes that copy different from the others. Consequently, caching and replication leads to **consistency** problems.

To what extent inconsistencies can be tolerated depends highly on the usage of a resource. For example, many Web users find it acceptable that their browser returns a cached document of which the validity has not been checked for the last few minutes. However, there are also many cases in which strong consistency guarantees need to be met, such as in the case of electronic stock exchanges and auctions. The problem with strong consistency is that an update must be immediately propagated to all other copies. Moreover, if two updates happen concurrently, it is often also required that each copy is updated in the same order. Situations such as these generally require some global synchronization mechanism. Unfortunately, such mechanisms are extremely hard or even impossible to implement in a scalable way, as she insists that photons and electrical signals obey a speed limit of 187 miles/msec (the speed of light). Consequently, scaling by replication may introduce other, inherently nonscalable solutions. We return to replication and consistency in Chapter 7.

When considering these scaling techniques, one could argue that size scalability is the least problematic from a technical point of view. In many cases, simply increasing the capacity of a machine will the save the day (at least temporarily and perhaps at significant costs). Geographical scalability is a much tougher problem as Mother Nature is getting in our way. Nevertheless, practice shows that combining distribution, replication, and caching techniques with different forms of consistency will often prove sufficient in many cases. Finally, administrative scalability seems to be the most difficult one, partly also because we need to solve nontechnical problems (e.g., politics of organizations and human collaboration). Nevertheless, progress has been made in this area, by simply *ignoring* administrative domains. The introduction and now widespread use of peer-to-peer technology demonstrates what can be achieved if end users simply take over control [Aberer and Hauswirth, 2005], [Lua et al., 2005], [Oram, 2001]. However, let it be clear that peer-to-peer technology can at best be only a partial solution to solving administrative scalability. Eventually, it will have to be dealt with.

1.2.5 Pitfalls

It should be clear by now that developing distributed systems can be a formidable task. As we will see many times throughout this book, there are so many issues to consider at the same time that it seems that only complexity can be the result. Nevertheless, by following a number of design principles, distributed systems can be developed that strongly adhere to the goals we set out in this chapter. Many principles follow the basic rules of decent software engineering and will not be repeated here.

However, distributed systems differ from traditional software because components are dispersed across a network. Not taking this dispersion into account during design time is what makes so many systems needlessly complex and results in mistakes that need to be patched later on. Peter Deutsch, then at Sun Microsystems, formulated these mistakes as the following false assumptions that everyone makes when developing a distributed application for the first time:

1. The network is reliable.
2. The network is secure.
3. The network is homogeneous.
4. The topology does not change.
5. Latency is zero.
6. Bandwidth is infinite.
7. Transport cost is zero.
8. There is one administrator.

Note how these assumptions relate to properties that are unique to distributed systems: reliability, security, heterogeneity, and topology of the network; latency and bandwidth; transport costs; and finally administrative domains. When developing nondistributed applications, many of these issues will most likely not show up.

Most of the principles we discuss in this book relate immediately to these assumptions. In all cases, we will be discussing solutions to problems that are caused by the fact that one or more assumptions are false. For example, reliable networks simply do not exist, leading to the impossibility of achieving failure transparency. We devote an entire chapter to deal with the fact that networked communication is inherently insecure. We have already argued that distributed systems need to take heterogeneity into account. In a similar vein, when discussing replication for solving scalability problems, we are essentially tackling latency and bandwidth problems. We will also touch upon management issues at various points throughout this book, dealing with the false assumptions of zero-cost transportation and a single administrative domain.

1.3 Types of distributed systems

Before starting to discuss the principles of distributed systems, let us first take a closer look at the various types of distributed systems. In the following we make a distinction between distributed computing systems, distributed information systems, and distributed embedded systems.

1.3.1 Distributed computing systems

An important class of distributed systems is the one for high-performance computing tasks. Roughly speaking, one can make a distinction between two subgroups. In **cluster computing** the underlying hardware consists of a collection of similar workstations or PCs, closely connected by means of a high-speed local-area network. In addition, each node runs the same operating system.

The situation becomes quite different in the case of **grid computing**. This subgroup consists of distributed systems that are often constructed as a federation of computer systems, where each system may fall under a different administrative domain, and may be very different when it comes to hardware, software, and deployed network technology.

Cluster computing systems

Cluster computing systems became popular when the price/performance ratio of personal computers and workstations improved. At a certain point, it became financially and technically attractive to build a supercomputer using off-the-shelf technology by simply hooking up a collection of relatively simple computers in a high-speed network. In virtually all cases, cluster computing is used for parallel programming in which a single (compute intensive) program is run in parallel on multiple machines.

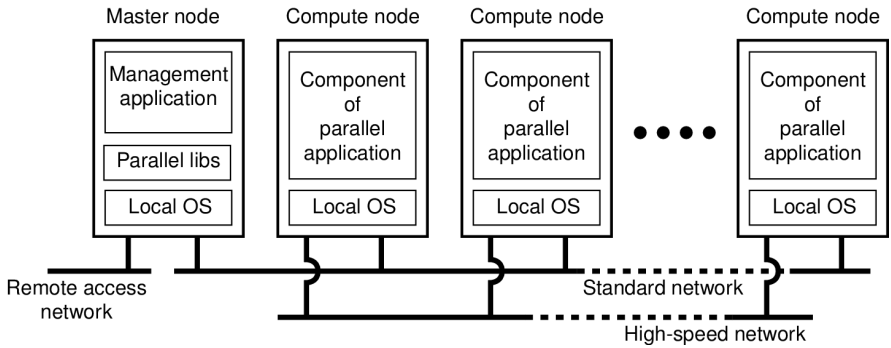

Figure 1.6: An example of a cluster computing system.

One well-known example of a cluster computer is formed by Linux-based Beowulf clusters, of which the general configuration is shown in Figure 1.6. Each cluster consists of a collection of compute nodes that are controlled and accessed by means of a single master node. The master typically handles the allocation of nodes to a particular parallel program, maintains a batch queue of submitted jobs, and provides an interface for the users of the system. As such, the master actually runs the middleware needed for the execution of programs and management of the cluster, while the compute nodes often need nothing else but a standard operating system.

An important part of this middleware is formed by the libraries for executing parallel programs. As we will discuss in Chapter 4, many of these libraries effectively provide only advanced message-based communication facilities, but are not capable of handling faulty processes, security, etc.

As an alternative to this hierarchical organization, a symmetric approach is followed in the MOSIX system [Amar et al., 2004]. MOSIX attempts to provide a **single-system image** of a cluster, meaning that to a process a cluster computer offers the ultimate distribution transparency by appearing to be a single computer. As we mentioned, providing such an image under all circumstances is impossible. In the case of MOSIX, the high degree

of transparency is provided by allowing processes to dynamically and preemptively migrate between the nodes that make up the cluster. Process migration allows a user to start an application on any node (referred to as the home node), after which it can transparently move to other nodes, for example, to make efficient use of resources. We will return to process migration in Chapter 3.

Grid computing systems

A characteristic feature of cluster computing is its homogeneity. In most cases, the computers in a cluster are largely the same, they all have the same operating system, and are all connected through the same network. In contrast, grid computing systems have a high degree of heterogeneity: no assumptions are made concerning hardware, operating systems, networks, administrative domains, security policies, etc.

A key issue in a grid computing system is that resources from different organizations are brought together to allow the collaboration of a group of people or institutions. Such a collaboration is realized in the form of a **virtual organization**. The people belonging to the same virtual organization have access rights to the resources that are provided to that organization. Typically, resources consist of compute servers (including supercomputers, possibly implemented as cluster computers), storage facilities, and databases. In addition, special networked devices such as telescopes, sensors, etc., can be provided as well.

Given its nature, much of the software for realizing grid computing evolves around providing access to resources from different administrative domains, and to only those users and applications that belong to a specific virtual organization. For this reason, focus is often on architectural issues. An architecture proposed by Foster et al. [2001] is shown in Figure 1.7.

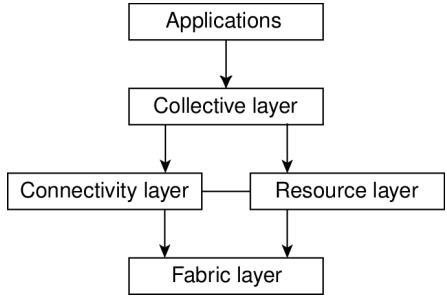

Figure 1.7: A layered architecture for grid computing systems.

The architecture consists of four layers. The lowest *fabric layer* provides interfaces to local resources at a specific site. Note that these interfaces are tailored to allow sharing of resources within a virtual organization. Typically, they will provide functions for querying the state and capabilities of a resource, along with functions for actual resource management (e.g., locking resources).

The *connectivity layer* consists of communication protocols for supporting grid transactions that span the usage of multiple resources. For example, protocols are needed to transfer data between resources, or to simply access a resource from a remote location. In addition, the connectivity layer will contain security protocols to authenticate users and resources. Note that in many cases human users are not authenticated; instead, programs acting on behalf of the users are authenticated. In this sense, delegating rights from a user to programs is an important function that needs to be supported in the connectivity layer. We return extensively to delegation when discussing security in distributed systems.

The *resource layer* is responsible for managing a single resource. It uses the functions provided by the connectivity layer and calls directly the interfaces made available by the fabric layer. For example, this layer will offer functions for obtaining configuration information on a specific resource, or, in general, to perform specific operations such as creating a process or reading data. The resource layer is thus seen to be responsible for access control, and hence will rely on the authentication performed as part of the connectivity layer.

The next layer in the hierarchy is the *collective layer*. It deals with handling access to multiple resources and typically consists of services for resource discovery, allocation and scheduling of tasks onto multiple resources, data replication, and so on. Unlike the connectivity and resource layer, which consist of a relatively small, standard collection of protocols, the collective layer may consist of many different protocols for many different purposes, reflecting the broad spectrum of services it may offer to a virtual organization.

Finally, the *application layer* consists of the applications that operate within a virtual organization and which make use of the grid computing environment.

Typically the collective, connectivity, and resource layer form the heart of what could be called a grid middleware layer. These layers jointly provide access to and management of resources that are potentially dispersed across multiple sites. An important observation from a middleware perspective is that with grid computing the notion of a site (or administrative unit) is common. This prevalence is emphasized by the gradual shift toward

a **service-oriented architecture** in which sites offer access to the various layers through a collection of Web services [Joseph et al., 2004]. This, by now, has led to the definition of an alternative architecture known as the **Open Grid Services Architecture** (**OGSA**). This architecture consists of various layers and many components, making it rather complex. Complexity seems to be the fate of any standardization process. Details on OGSA can be found in Foster et al. [2006].

1.3.2 Distributed information systems

Another important class of distributed systems is found in organizations that were confronted with a wealth of networked applications, but for which interoperability turned out to be a painful experience. Many of the existing middleware solutions are the result of working with an infrastructure in which it was easier to integrate applications into an enterprise-wide information system [Alonso et al., 2004], [Bernstein, 1996].

We can distinguish several levels at which integration took place. In many cases, a networked application simply consisted of a server running that application (often including a database) and making it available to remote programs, called **clients**. Such clients could send a request to the server for executing a specific operation, after which a response would be sent back. Integration at the lowest level would allow clients to wrap a number of requests, possibly for different servers, into a single larger request and have it executed as a **distributed transaction**. The key idea was that all, or none of the requests would be executed.

As applications became more sophisticated and were gradually separated into independent components (notably distinguishing database components from processing components), it became clear that integration should also take place by letting applications communicate directly with each other. This has now led to a huge industry that concentrates on **enterprise application integration** (**EAI**). In the following, we concentrate on these two forms of distributed systems.

Transaction processing systems

To clarify our discussion, let us concentrate on database applications. In practice, operations on a database are usually carried out in the form of **transactions**. Programming using transactions requires special primitives that must either be supplied by the underlying distributed system or by the language runtime system. Typical examples of transaction primitives are shown in Figure 1.8. The exact list of primitives depends on what kinds of objects are being used in the transaction [Gray and Reuter, 1993]. In a

mail system, there might be primitives to send, receive, and forward mail. In an accounting system, they might be quite different. READ and WRITE are typical examples, however. Ordinary statements, procedure calls, and so on, are also allowed inside a transaction. In particular, we mention that remote procedure calls (RPCs), that is, procedure calls to remote servers, are often also encapsulated in a transaction, leading to what is known as a **transactional RPC**. We discuss RPCs extensively in Chapter 4.

Primitive	Description
BEGIN_TRANSACTION	Mark the start of a transaction
END_TRANSACTION	Terminate the transaction and try to commit
ABORT_TRANSACTION	Kill the transaction and restore the old values
READ	Read data from a file, a table, or otherwise
WRITE	Write data to a file, a table, or otherwise

Figure 1.8: Example primitives for transactions.

BEGIN_TRANSACTION and END_TRANSACTION are used to delimit the scope of a transaction. The operations between them form the body of the transaction. The characteristic feature of a transaction is either all of these operations are executed or none are executed. These may be system calls, library procedures, or bracketing statements in a language, depending on the implementation.

This all-or-nothing property of transactions is one of the four characteristic properties that transactions have. More specifically, transactions are:

1. Atomic: To the outside world, the transaction happens indivisibly.
2. Consistent: The transaction does not violate system invariants.
3. Isolated: Concurrent transactions do not interfere with each other.
4. Durable: Once a transaction commits, the changes are permanent.

These properties are often referred to by their initial letters: **ACID**.

The first key property exhibited by all transactions is that they are **atomic**. This property ensures that each transaction either happens completely, or not at all, and if it happens, it happens in a single indivisible, instantaneous action. While a transaction is in progress, other processes (whether or not they are themselves involved in transactions) cannot see any of the intermediate states.

The second property says that they are **consistent**. What this means is that if the system has certain invariants that must always hold, if they held before the transaction, they will hold afterward too. For example, in a

banking system, a key invariant is the law of conservation of money. After every internal transfer, the amount of money in the bank must be the same as it was before the transfer, but for a brief moment during the transaction, this invariant may be violated. The violation is not visible outside the transaction, however.

The third property says that transactions are **isolated** or **serializable**. What it means is that if two or more transactions are running at the same time, to each of them and to other processes, the final result looks as though all transactions ran sequentially in some (system dependent) order.

The fourth property says that transactions are **durable**. It refers to the fact that once a transaction commits, no matter what happens, the transaction goes forward and the results become permanent. No failure after the commit can undo the results or cause them to be lost. (Durability is discussed extensively in Chapter 8.)

So far, transactions have been defined on a single database. A **nested transaction** is constructed from a number of subtransactions, as shown in Figure 1.9. The top-level transaction may fork off children that run in parallel with one another, on different machines, to gain performance or simplify programming. Each of these children may also execute one or more subtransactions, or fork off its own children.

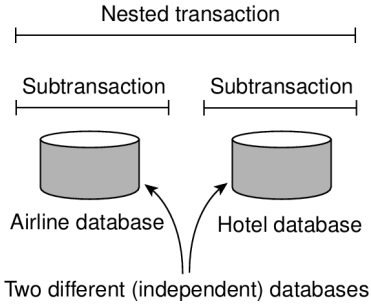

Figure 1.9: A nested transaction.

Subtransactions give rise to a subtle, but important, problem. Imagine that a transaction starts several subtransactions in parallel, and one of these commits, making its results visible to the parent transaction. After further computation, the parent aborts, restoring the entire system to the state it had before the top-level transaction started. Consequently, the results of the subtransaction that committed must nevertheless be undone. Thus the permanence referred to above applies only to top-level transactions.

Since transactions can be nested arbitrarily deeply, considerable administration is needed to get everything right. The semantics are clear, however.

When any transaction or subtransaction starts, it is conceptually given a private copy of all data in the entire system for it to manipulate as it wishes. If it aborts, its private universe just vanishes, as if it had never existed. If it commits, its private universe replaces the parent's universe. Thus if a subtransaction commits and then later a new subtransaction is started, the second one sees the results produced by the first one. Likewise, if an enclosing (higher-level) transaction aborts, all its underlying subtransactions have to be aborted as well.

Nested transactions are important in distributed systems, for they provide a natural way of distributing a transaction across multiple machines. They follow a *logical* division of the work of the original transaction. For example, a transaction for planning a trip by which three different flights need to be reserved can be logically split up into three subtransactions. Each of these subtransactions can be managed separately and independent of the other two.

In the early days of enterprise middleware systems, the component that handled distributed (or nested) transactions formed the core for integrating applications at the server or database level. This component was called a **transaction processing monitor** or **TP monitor** for short. Its main task was to allow an application to access multiple server/databases by offering it a transactional programming model, as shown in Figure 1.10.

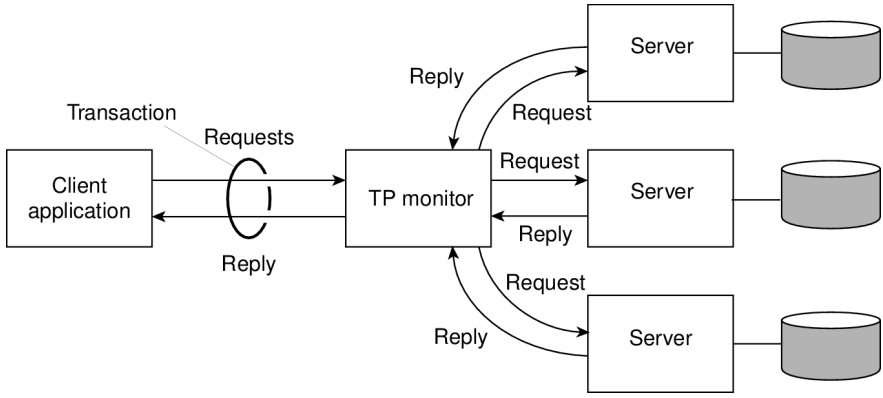

Figure 1.10: The role of a TP monitor in distributed systems.

Enterprise application integration

As mentioned, the more applications became decoupled from the databases they were built upon, the more evident it became that facilities were needed

to integrate applications independent from their databases. In particular, application components should be able to communicate directly with each other and not merely by means of the request/reply behavior that was supported by transaction processing systems.

This need for interapplication communication led to many different communication models, which we will discuss in detail in this book (and for which reason we shall keep it brief for now). The main idea was that existing applications could directly exchange information, as shown in Figure 1.11.

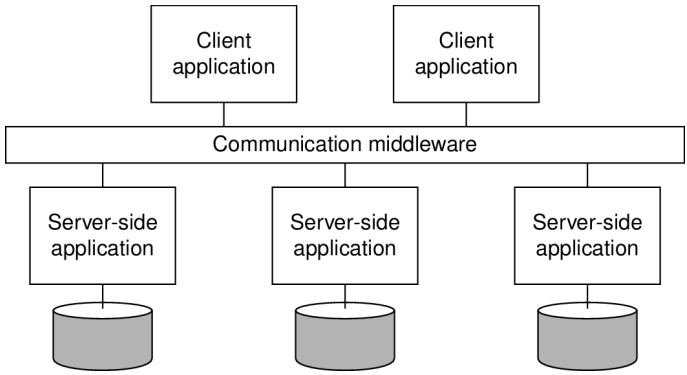

Figure 1.11: Middleware as a communication facilitator in enterprise application integration.

Several types of communication middleware exist. With **remote procedure calls** (**RPC**), an application component can effectively send a request to another application component by doing a local procedure call, which results in the request being packaged as a message and sent to the callee. Likewise, the result will be sent back and returned to the application as the result of the procedure call.

As the popularity of object technology increased, techniques were developed to allow calls to remote objects, leading to what is known as **remote method invocations** (**RMI**). An RMI is essentially the same as an RPC, except that it operates on objects instead of applications.

RPC and RMI have the disadvantage that the caller and callee both need to be up and running at the time of communication. In addition, they need to know exactly how to refer to each other. This tight coupling is often experienced as a serious drawback, and has led to what is known as **message-oriented middleware**, or simply **MOM**. In this case, applications simply send messages to logical contact points, often described by means of a subject. Likewise, applications can indicate their interest for a specific type of message, after which the communication middleware will take

care that those messages are delivered to those applications. These so-called **publish/subscribe** systems form an important and expanding class of distributed systems. We will discuss them at length in Chapter 13.

1.3.3 Distributed pervasive systems

The distributed systems we have been discussing so far are largely characterized by their stability: nodes are fixed and have a more or less permanent and high-quality connection to a network. To a certain extent, this stability has been realized through the various techniques that are discussed in this book and which aim at achieving distribution transparency. For example, the wealth of techniques for masking failures and recovery will give the impression that only occasionally things may go wrong. Likewise, we have been able to hide aspects related to the actual network location of a node, effectively allowing users and applications to believe that nodes stay put.

However, matters have become very different with the introduction of mobile and embedded computing devices. We are now confronted with distributed systems in which instability is the default behavior. The devices in these, what we refer to as **distributed pervasive systems**, are often characterized by being small, battery-powered, mobile, and having only a wireless connection, although not all these characteristics apply to all devices. Moreover, these characteristics need not necessarily be interpreted as restrictive, as is illustrated by the possibilities of modern smart phones [Roussos et al., 2005].

As its name suggests, a distributed pervasive system is part of our surroundings (and as such, is generally inherently distributed). An important feature is the general lack of human administrative control. At best, devices can be configured by their owners, but otherwise they need to automatically discover their environment and "nestle in" as best as possible. This nestling in has been made more precise by Grimm et al. [2004]. by formulating the following three requirements for pervasive applications:

1. Embrace contextual changes.
2. Encourage ad hoc composition.
3. Recognize sharing as the default.

Embracing contextual changes means that a device must be continuously be aware of the fact that its environment may change all the time. One of the simplest changes is discovering that a network is no longer available, for example, because a user is moving between base stations. In such a case, the application should react, possibly by automatically connecting to another network, or taking other appropriate actions.

Encouraging ad hoc composition refers to the fact that many devices in pervasive systems will be used in very different ways by different users. As a result, it should be easy to configure the suite of applications running on a device, either by the user or through automated (but controlled) interposition.

One very important aspect of pervasive systems is that devices generally join the system in order to access (and possibly provide) information. This calls for means to easily read, store, manage, and share information. In light of the intermittent and changing connectivity of devices, the space where accessible information resides will most likely change all the time.

Mascolo et al. [2004] as well as Niemela and Latvakoski [2004] came to similar conclusions: in the presence of mobility, devices should support easy and application-dependent adaptation to their local environment. They should be able to efficiently discover services and react accordingly. It should be clear from these requirements that distribution transparency is not really in place in pervasive systems. In fact, distribution of data, processes, and control is *inherent* to these systems, for which reason it may be better just to simply expose it rather than trying to hide it. Let us now take a look at some concrete examples of pervasive systems.

Home systems

An increasingly popular type of pervasive system, but which may perhaps be the least constrained, are systems built around home networks. These systems generally consist of one or more personal computers, but more importantly integrate typical consumer electronics such as TVs, audio and video equipment, gaming devices, (smart) phones, PDAs, and other personal wearables into a single system. In addition, we can expect that all kinds of devices such as kitchen appliances, surveillance cameras, clocks, controllers for lighting, and so on, will all be hooked up into a single distributed system.

From a system's perspective there are several challenges that need to be addressed before pervasive home systems become reality. An important one is that such a system should be completely self-configuring and self-managing. It cannot be expected that end users are willing and able to keep a distributed home system up and running if its components are prone to errors (as is the case with many of today's devices.) Much has already been accomplished through the **Universal Plug and Play** (**UPnP**) standards by which devices automatically obtain IP addresses, can discover each other, etc. [UPnP Forum, 2003]. However, more is needed. For example, it is unclear how software and firmware in devices can be easily updated without

manual intervention, or when updates do take place, that compatibility with other devices is not violated.

Another pressing issue is managing what is known as a *"personal space."* Recognizing that a home system consists of many shared as well as personal devices, and that the data in a home system is also subject to sharing restrictions, much attention is paid to realizing such personal spaces. For example, part of Alice's personal space may consist of her agenda, family photo's, a diary, music and videos that she bought, etc. These personal assets should be stored in such a way that Alice has access to them whenever appropriate. Moreover, parts of this personal space should be (temporarily) accessible to others, for example, when she needs to make a business appointment.

Fortunately, things may become simpler. It has long been thought that the personal spaces related to home systems were inherently distributed across the various devices. Obviously, such a dispersion can easily lead to significant synchronization problems. However, problems may be alleviated due to the rapid increase in the capacity of hard disks, along with a decrease in their size. Configuring a multi-terabyte storage unit for a personal computer is not really a problem. At the same time, portable hard disks having a capacity of hundreds of gigabytes are being placed inside relatively small portable media players. With these continuously increasing capacities, we may see pervasive home systems adopt an architecture in which a single machine acts as a master (and is hidden away somewhere in the basement next to the central heating), and all other fixed devices simply provide a convenient interface for humans. Personal devices will then be crammed with daily needed information, but will never run out of storage.

However, having enough storage does not solve the problem of managing personal spaces. Being able to store huge amounts of data shifts the problem to storing *relevant* data and being able to find it later. Increasingly we will see pervasive systems, like home networks, equipped with what are called **recommenders**, programs that consult what other users have stored in order to identify similar taste, and from that subsequently derive which content to place in one's personal space. An interesting observation is that the amount of information that recommender programs need to do their work is often small enough to allow them to be run on PDAs [Miller et al., 2004].

Electronic health care systems

Another important and upcoming class of pervasive systems are those related to (personal) electronic health care. With the increasing cost of medical treatment, new devices are being developed to monitor the well-being of individuals and to automatically contact physicians when needed.

In many of these systems, a major goal is to prevent people from being hospitalized.

Personal health care systems are often equipped with various sensors organized in a (preferably wireless) body-area network (BAN). An important issue is that such a network should at worst only minimally hinder a person. To this end, the network should be able to operate while a person is moving, with no strings (i.e., wires) attached to immobile devices.

This requirement leads to two obvious organizations, as shown in Figure 1.12. In the first one, a central hub is part of the BAN and collects data as needed. From time to time, this data is then offloaded to a larger storage device. The advantage of this scheme is that the hub can also manage the BAN. In the second scenario, the BAN is continuously hooked up to an external network, again through a wireless connection, to which it sends monitored data. Separate techniques will need to be deployed for managing the BAN. Of course, further connections to a physician or other people may exist as well.

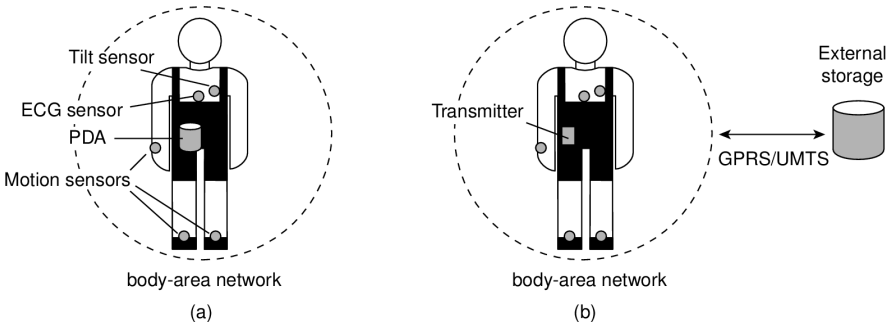

Figure 1.12: Monitoring a person in a pervasive electronic health care system, using (a) a local hub or (b) a continuous wireless connection.

From a distributed system's perspective we are immediately confronted with questions such as:

1. Where and how should monitored data be stored?
2. How can we prevent loss of crucial data?
3. What infrastructure is needed to generate and propagate alerts?
4. How can physicians provide online feedback?
5. How can extreme robustness of the monitoring system be realized?
6. What are the security issues and how can the proper policies be .br enforced?

Unlike home systems, we cannot expect the architecture of pervasive health care systems to move toward single-server systems and have the monitoring devices operate with minimal functionality. On the contrary: for reasons of efficiency, devices and body-area networks will be required to support **in-network data processing**, meaning that monitoring data will, for example, have to be aggregated before permanently storing it or sending it to a physician. Unlike the case for distributed information systems, there is yet no clear answer to these questions.

Sensor networks

Our last example of pervasive systems is sensor networks. These networks in many cases form part of the enabling technology for pervasiveness and we see that many solutions for sensor networks return in pervasive applications. What makes sensor networks interesting from a distributed system's perspective is that in virtually all cases they are used for processing information. In this sense, they do more than just provide communication services, which is what traditional computer networks are all about. Akyildiz et al. [2002] provide an overview from a networking perspective. A more systems-oriented introduction to sensor networks is given by Zhao and Guibas [2004]. Strongly related are **mesh networks** which essentially form a collection of (fixed) nodes that communicate through wireless links. These networks may form the basis for many medium-scale distributed systems. An overview is provided in Akyildiz et al. [2005].

A sensor network typically consists of tens to hundreds or thousands of relatively small nodes, each equipped with a sensing device. Most sensor networks use wireless communication, and the nodes are often battery powered. Their limited resources, restricted communication capabilities, and constrained power consumption demand that efficiency be high on the list of design criteria.

The relation with distributed systems can be made clear by considering sensor networks as distributed databases. This view is quite common and easy to understand when realizing that many sensor networks are deployed for measurement and surveillance applications [Bonnet et al., 2002]. In these cases, an operator would like to extract information from (a part of) the network by simply issuing queries such as "What is the northbound traffic load on Highway 1?" Such queries resemble those of traditional databases. In this case, the answer will probably need to be provided through collaboration of many sensors located around Highway 1, while leaving other sensors untouched.

To organize a sensor network as a distributed database, there are essentially two extremes, as shown in Figure 1.13. First, sensors do not cooperate

but simply send their data to a centralized database located at the operator's site. The other extreme is to forward queries to relevant sensors and to let each compute an answer, requiring the operator to sensibly aggregate the returned answers.

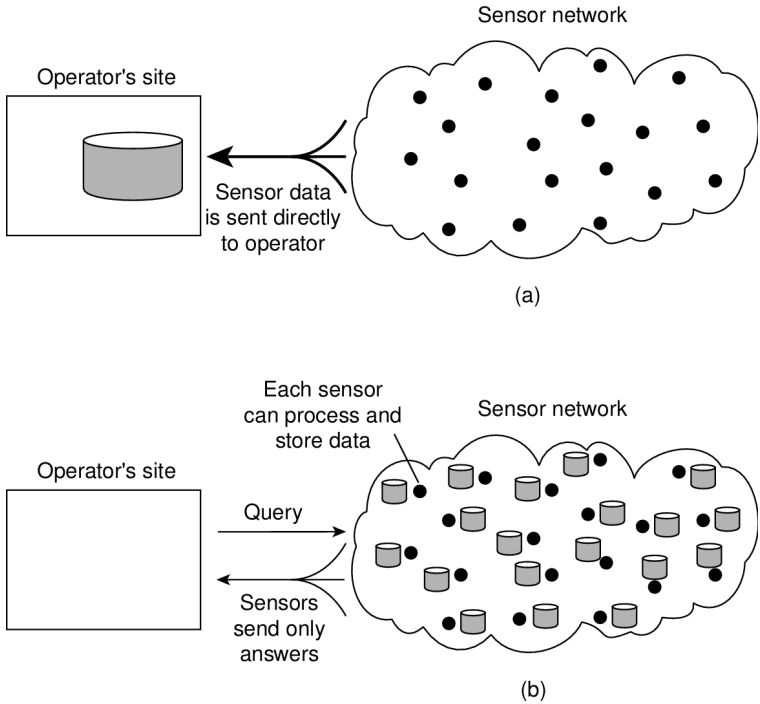

Figure 1.13: Organizing a sensor network database, while storing and processing data (a) only at the operator's site or (b) only at the sensors.

Neither of these solutions is very attractive. The first one requires that sensors send all their measured data through the network, which may waste network resources and energy. The second solution may also be wasteful as it discards the aggregation capabilities of sensors which would allow much less data to be returned to the operator. What is needed are facilities for **in-network data processing**, as we also encountered in pervasive health care systems.

In-network processing can be done in numerous ways. One obvious one is to forward a query to all sensor nodes along a tree encompassing all nodes and to subsequently aggregate the results as they are propagated back to the root, where the initiator is located. Aggregation will take place where two or more branches of the tree come to together. As simple as this scheme may sound, it introduces difficult questions:

1. How do we (dynamically) set up an efficient tree in a sensor network?
2. How does aggregation of results take place? Can it be controlled?
3. What happens when network links fail?

These questions have been partly addressed in TinyDB, which implements a declarative (database) interface to wireless sensor networks. In essence, TinyDB can use any tree-based routing algorithm. An intermediate node will collect and aggregate the results from its children, along with its own findings, and send that toward the root. To make matters efficient, queries span a period of time allowing for careful scheduling of operations so that network resources and energy are optimally consumed. Details can be found in Madden et al. [2005].

However, when queries can be initiated from different points in the network, using single-rooted trees such as in TinyDB may not be efficient enough. As an alternative, sensor networks may be equipped with special nodes where results are forwarded to, as well as the queries related to those results. To give a simple example, queries and results related temperature readings are collected at a different location than those related to humidity measurements. This approach corresponds directly to the notion of publish/subscribe systems, which we will discuss extensively in Chapter 13.

1.4 Summary

Distributed systems consist of autonomous computers that work together to give the appearance of a single coherent system. One important advantage is that they make it easier to integrate different applications running on different computers into a single system. Another advantage is that when properly designed, distributed systems scale well with respect to the size of the underlying network. These advantages often come at the cost of more complex software, degradation of performance, and also often weaker security. Nevertheless, there is considerable interest worldwide in building and installing distributed systems.

Distributed systems often aim at hiding many of the intricacies related to the distribution of processes, data, and control. However, this distribution transparency not only comes at a performance price, but in practical situations it can never be fully achieved. The fact that trade-offs need to be made between achieving various forms of distribution transparency is inherent to the design of distributed systems, and can easily complicate their understanding.

Matters are further complicated by the fact that many developers initially make assumptions about the underlying network that are fundamentally wrong. Later, when assumptions are dropped, it may turn out to be difficult

to mask unwanted behavior. A typical example is assuming that network latency is not significant. Later, when porting an existing system to a wide-area network, hiding latencies may deeply affect the system's original design. Other pitfalls include assuming that the network is reliable, static, secure, and homogeneous.

Different types of distributed systems exist which can be classified as being oriented toward supporting computations, information processing, and pervasiveness. Distributed computing systems are typically deployed for high-performance applications often originating from the field of parallel computing. A huge class of distributed can be found in traditional office environments where we see databases playing an important role. Typically, transaction processing systems are deployed in these environments. Finally, an emerging class of distributed systems is where components are small and the system is composed in an ad hoc fashion, but most of all is no longer managed through a system administrator. This last class is typically represented by ubiquitous computing environments.

Bibliography

Abadi M. and Needham R. Prudent Engineering Practice for Cryptographic Protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, Jan. 1996. Cited on 436

Abdullahi S. and Ringwood G. Garbage Collecting the Internet: A Survey of Distributed Garbage Collection. *ACM Computing Surveys*, 30(3):330–373, Sept. 1998. Cited on 212

Aberer K. and Hauswirth M. Peer-to-Peer Systems. In Singh M., editor, *The Practical Handbook of Internet Computing*, chapter 35. CRC Press, Boca Raton, FL, 2005. Cited on

Aberer K., Alima L. O., Ghodsi A., Girdzijauskas S., Hauswirth M., and Haridi S. The Essence of P2P: A Reference Architecture for Overlay Networks. In *5th International Conference on Peer-to-Peer Computing*, pages 11–20, Los Alamitos, CA., Aug. 2005. IEEE, IEEE Computer Society Press. Cited on 63

Adar E. and Huberman B. A. Free Riding on Gnutella. Hewlett Packard, Information Dynamics Lab, Jan. 2000. Cited on 72

Aiyer A., Alvisi L., Clement A., Dahlin M., and Martin J.-P. BAR Fault Tolerance for Cooperative Services. In *20th Symposium on Operating System Principles*, pages 45–58, New York, NY, Oct. 2005. ACM, ACM Press. Cited on 368

Akyildiz I. F., Su W., Sankarasubramaniam Y., and Cayirci E. A Survey on Sensor Networks. *IEEE Communications Magazine*, 40(8):102–114, Aug. 2002. Cited on

Akyildiz I. F., Wang X., and Wang W. Wireless Mesh Networks: A Survey. *Computer Networks*, 47(4):445–487, Mar. 2005. Cited on

Albitz P. and Liu C. *DNS and BIND*. O'Reilly & Associates, Sebastopol, CA., 4th edition, 2001. Cited on 605

Allen R. and Lowe-Norris A. *Windows 2000 Active Directory*. O'Reilly & Associates, Sebastopol, CA., 2nd edition, 2003. Cited on 250

Allman M. An Evaluation of XML-RPC. *Performace Evaluation Review*, 30(4):2–11, Mar. 2003. Cited on 612

Alonso G., Casati F., Kuno H., and Machiraju V. *Web Services: Concepts, Architectures and Applications*. Springer-Verlag, Berlin, 2004. Cited on 595, 599

Alvisi L. and Marzullo K. Message Logging: Pessimistic, Optimistic, Causal, and Optimal. *IEEE Transactions on Software Engineering*, 24(2):149–159, Feb. 1998. Cited

on 405, 406

Amar L., Barak A., and Shiloh A. The MOSIX Direct File System Access Method for Supporting Scalable Cluster File Systems. *Cluster Computing*, 7(2):141–150, Apr. 2004. Cited on

Anderson O. T., Luan L., Everhart C., Pereira M., Sarkar R., and Xu J. Global Namespace for Files. *IBM Systems Journal*, 43(4):702–722, Apr. 2004. Cited on 554

Anderson T., Bershad B., Lazowska E., and Levy H. Scheduler Activations: Efficient Kernel Support for the User-Level Management of Parallelism. In *13th Symposium on Operating System Principles*, pages 95–109, New York, NY, Oct. 1991. ACM, ACM Press. Cited on 95

Andrews G. *Foundations of Multithreaded, Parallel, and Distributed Programming*. Addison-Wesley, Reading, MA., 2000. Cited on 259

Androutsellis-Theotokis S. and Spinellis D. A Survey of Peer-to-Peer Content Distribution Technologies. *ACM Computing Surveys*, 36(4):335–371, Dec. 2004. Cited on 63

Araujo F. and Rodrigues L. Survey on Position-Based Routing. Technical Report MINEMA TR-01, University of Lisbon, Oct. 2005. Cited on 291

Arkills B. *LDAP Directories Explained: An Introduction and Analysis*. Addison-Wesley, Reading, MA., 2003. Cited on 247

Aron M., Sanders D., Druschel P., and Zwaenepoel W. Scalable Content-aware Request Distribution in Cluster-based Network Servers. In *USENIX Annual Technical Conference*, pages 323–336, San Diego, CA, June 2000. USENIX. Cited on 604

Attiya H. and Welch J. *Distributed Computing Fundamentals, Simulations, and Advanced Topics*. John Wiley, New York, 2nd edition, 2004. Cited on 259

Avizienis A., Laprie J.-C., Randell B., and Landwehr C. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, Jan. 2004. Cited on 355

Awadallah A. and Rosenblum M. The vMatrix: A Network of Virtual Machine Monitors for Dynamic Content Distribution. In *7th Web Caching Workshop*, Aug. 2002. Cited on 101

Awadallah A. and Rosenblum M. The vMatrix: Server Switching. In *10th Workshop on Future Trends in Distributed Computing Systems*, pages 110–118, Los Alamitos, CA., May 2004. IEEE, IEEE Computer Society Press. Cited on 116

Babaoglu O. and Toueg S. Non-Blocking Atomic Commitment. In Mullender S., editor, *Distributed Systems*, pages 147–168. Addison-Wesley, Wokingham, 2nd edition, 1993. Cited on 395

Babaoglu O., Bartoli A., Maverick V., Patarin S., Vučković J., and Wu H. A Framework for Prototyping J2EE Replication Algorithms. In *International Symposium on Distributed Objects and Applications (DOA)*, Lecture Notes in Computer Science, Berlin, Oct. 2004. Springer-Verlag. Cited on 514

Babaoglu O., Jelasity M., Montresor A., Fetzer C., Leonardi S., Moorsel A.van , and Steen M.van , editors. *Self-star Properties in Complex Information Systems*, volume 3460 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2005. Cited on 79

Babcock B., Babu S., Datar M., Motwani R., and Widom J. Models and Issues in Data Stream Systems. In *21st Symposium on Principles of Distributed Computing*, pages 1–16, New York, NY, July 2002. ACM, ACM Press. Cited on 184

Bal H. *The Shared Data-Object Model as a Paradigm for Programming Distributed Systems*. PhD., Vrije Universiteit, Amsterdam, 1989. Cited on 487

Balakrishnan H., Kaashoek M. F., Karger D., Morris R., and Stoica I. Looking up Data in P2P Systems. *Communications of the ACM*, 46(2):43–48, Feb. 2003. Cited on 63, 214

Balazinska M., Balakrishnan H., and Karger D. INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery. In *1st International Conference on Pervasive Computing*, volume 2414 of *Lecture Notes in Computer Science*, pages 195–210, Berlin, Aug. 2002. Springer-Verlag. Cited on 251, 252

Ballintijn G. *Locating Objects in a Wide-area System*. PhD thesis, Vrije Universiteit Amsterdam, 2003. Cited on 218, 526

Baratto R. A., Nieh J., and Kim L. THINC: A Remote Display Architecture for Thin-Client Computing. In *20th Symposium on Operating System Principles*, pages 277–290, New York, NY, Oct. 2005. ACM, ACM Press. Cited on 107

Barborak M., Malek M., and Dahbura A. The Consensus Problem in Fault-Tolerant Computing. *ACM Computing Surveys*, 25(2):171–220, June 1993. Cited on 367

Barham P., Dragovic B., Fraser K., Hand S., Harris T., Ho A., Neugebar R., Pratt I., and Warfield A. Xen and the Art of Virtualization. In *19th Symposium on Operating System Principles*, pages 164–177, New York, NY, Oct. 2003. ACM, ACM Press. Cited on 102

Barker W. Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher. NIST Special Publication 800-67, May 2004. Cited on 428

Barron D. *Pascal – The Language and its Implementation*. John Wiley, New York, 1981. Cited on 133

Barroso L., Deam J., and Holze U. Web Search for a Planet: The Google Cluster Architecture. *IEEE Micro*, 23(2):21–28, Mar. 2003. Cited on 537

Baryshnikov Y., Coffman E. G., Pierre G., Rubenstein D., Squillante M., and Yimwadsana T. Predictability of Web-Server Traffic Congestion. In *10th Web Caching Workshop*, pages 97–103. IEEE, Sept. 2005. Cited on 623

Basile C., Whisnant K., Kalbarczyk Z., and Iyer R. K. Loose Synchronization of Multithreaded Replicas. In *21st Symposium on Reliable Distributed Systems*, pages 250–255, Los Alamitos, CA., 2002. IEEE, IEEE Computer Society Press. Cited on 514

Basile C., Kalbarczyk Z., and Iyer R. K. A Preemptive Deterministic Scheduling Algorithm for Multithreaded Replicas. In *International Conference on Dependable Systems and Networks*, pages 149–158, Los Alamitos, CA., June 2003. IEEE Computer Society Press. Cited on 514

Bass L., Clements P., and Kazman R. *Software Architecture in Practice*. Addison-Wesley, Reading, MA., 2nd edition, 2003. Cited on 52, 53, 54

Bavier A., Bowman M., Chun B., Culler D., Karlin S., Muir S., Peterson L., Roscoe T., Spalink T., and Wawrzoniak M. Operating System Support for Planetary-Scale Network Services. In *1st Symposium on Networked Systems Design and Implemen-*

tation, pages 245–266, Berkeley, CA, Mar. 2004. USENIX, USENIX. Cited on 121, 125

Berners-Lee T., Cailliau R., Nielson H. F., and Secret A. The World-Wide Web. *Communications of the ACM*, 37(8):76–82, Aug. 1994. Cited on 589

Berners-Lee T., Fielding R., and Masinter L. Uniform Resource Identifiers (URI): Generic Syntax. RFC 3986, Jan. 2005. Cited on 613

Bernstein P. Middleware: A Model for Distributed System Services. *Communications of the ACM*, 39(2):87–98, Feb. 1996. Cited on

Bernstein P., Hadzilacos V., and Goodman N. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, MA., 1987. Cited on 390, 398

Bershad B., Zekauskas M., and Sawdon W. The Midway Distributed Shared Memory System. In *COMPCON*, pages 528–537. IEEE, 1993. Cited on 317

Bertino E. and Ferrari E. Secure and Selective Dissemination of XML Documents. *ACM Transactions on Information and System Security*, 5(3):290–331, 2002. Cited on 666

Bhagwan R., Tati K., Cheng Y., Savage S., and Voelker G. M. Total Recall: Systems Support for Automated Availability Management . In *1st Symposium on Networked Systems Design and Implementation*, pages 337–350, Berkeley, CA, Mar. 2004. USENIX, USENIX. Cited on 575

Bharambe A. R., Agrawal M., and Seshan S. Mercury: Supporting Scalable Multi-Attribute Range Queries. In *SIGCOMM*, pages 353–366, New York, NY, Aug. 2004. ACM Press. Cited on 254, 646

Birman K. *Reliable Distributed Systems: Technologies, Web Services, and Applications*. Springer-Verlag, Berlin, 2005. Cited on 112, 368, 630

Birman K. A Response to Cheriton and Skeen's Criticism of Causal and Totally Ordered Communication. *Operating Systems Review*, 28(1):11–21, Jan. 1994. Cited on 280

Birman K. and Joseph T. Reliable Communication in the Presence of Failures. *ACM Transactions on Computer Systems*, 5(1):47–76, Feb. 1987. Cited on 384

Birman K. and Renesse R.van , editors. *Reliable Distributed Computing with the Isis Toolkit*. IEEE Computer Society Press, Los Alamitos, CA., 1994. Cited on 280

Birman K., Schiper A., and Stephenson P. Lightweight Causal and Atomic Group Multicast. *ACM Transactions on Computer Systems*, 9(3):272–314, Aug. 1991. Cited on 387

Birrell A. and Nelson B. Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems*, 2(1):39–59, Feb. 1984. Cited on 148

Bishop M. *Computer Security: Art and Science*. Addison-Wesley, Reading, MA., 2003. Cited on 420

Bjornson R. *Linda on Distributed Memory Multicomputers*. Ph.D., Yale University, Department of Computer Science, 1993. Cited on 656

Black A. and Artsy Y. Implementing Location Independent Invocation. *IEEE Transactions on Parallel and Distributed Systems*, 1(1):107–119, Jan. 1990. Cited on 212

Blair G. and Stefani J.-B. *Open Distributed Processing and Multimedia*. Addison-Wesley, Reading, MA., 1998. Cited on 191

Blair G., Coulson G., and Grace P. Research Directions in Reflective Middleware: the Lancaster Experience. In *3rd Workshop on Reflective and Adaptive Middleware*, pages 262–267, New York, NY, Oct. 2004. ACM, ACM Press. Cited on 78

Blake-Wilson S., Nystrom M., Hopwood D., Mikkelsen J., and Wright T. Transport Layer Security (TLS) Extensions. RFC 3546, June 2003. Cited on 631

Blaze M. *Caching in Large-Scale Distributed File Systems*. PhD thesis, Department of Computer Science, Princeton University, Jan. 1993. Cited on 333

Bonnet P., Gehrke J., and Seshadri P. Towards Sensor Database Systems. In *2nd International Conference on Mobile Data Management*, volume 1987 of *Lecture Notes in Computer Science*, pages 3–14, Berlin, Jan. 2002. Springer-Verlag. Cited on

Booth D., Haas H., McCabe F., Newcomer E., Champion M., Ferris C., and Orchard D. Web Services Architecture. W3C Working Group Note, Feb. 2004. Cited on 596

Bouchenak S., Boyer F., Hagimont D., Krakowiak S., Mos A., Palma3 N.de , Quema3 V., and Stefani J.-B. Architecture-Based Autonomous Repair Management: An Application to J2EE Clusters. In *24th Symposium on Reliable Distributed Systems*, pages 13–24, Los Alamitos, CA., Oct. 2005. IEEE, IEEE Computer Society Press. Cited on 86

Brewer E. Lessons from Giant-Scale Services. *IEEE Internet Computing*, 5(4):46–55, July 2001. Cited on 121

Bruneton E., Coupaye T., Leclercq M., Quema V., and Stefani J.-B. An Open Component Model and Its Support in Java. In *7th International Symposium on Component-based Software Engineering*, volume 3054 of *Lecture Notes in Computer Science*, pages 7–22, Berlin, 2004. Springer-Verlag. Cited on 86

Budhijara N., Marzullo K., Schneider F., and Toueg S. The Primary-Backup Approach. In Mullender S., editor, *Distributed Systems*, pages 199–216. Addison-Wesley, Wokingham, 2nd edition, 1993. Cited on 342

Budhiraja N. and Marzullo K. Tradeoffs in Implementing Primary-Backup Protocols. Technical Report TR 92-1307, Department of Computer Science, Cornell University, 1992. Cited on 342

Burns R. C., Rees R. M., Stockmeyer L. J., and Long D. D. E. Scalable Session Locking for a Distributed File System. *Cluster Computing*, 4(4):295–306, Oct. 2001. Cited on 560

Busi N., Montresor A., and Zavattaro G. Data-driven Coordination in Peer-to-Peer Information Systems. *International Journal on Cooperative Information Systems*, 13(1): 63–89, Mar. 2004. Cited on 643

Butt A. R., Johnson T. A., Zheng Y., and Hu Y. C. Kosha: A Peer-to-Peer Enhancement for the Network File System. In *International Conference on Supercomputing*, pages 51–61, Los Alamitos, CA., Nov. 2004. IEEE, IEEE Computer Society Press. Cited on 541

Cabri G., Leonardi L., and Zambonelli F. Mobile-Agent Coordination Models for Internet Applications. *Computer*, 33(2):82–89, Feb. 2000. Cited on 636

Cai M., Chervenak A., and Frank M. A Peer-to-Peer Replica Location Service Based on A Distributed Hash Table. In *High Performance Computing, Networking, and Storage Conference*, pages 56–67, New York, NY, Nov. 2004. ACM/IEEE, ACM Press. Cited on 572

Callaghan B. *NFS Illustrated*. Addison-Wesley, Reading, MA., 2000. Cited on 531, 552

Candea G., Brown A. B., Fox A., and Patterson D. Recovery-Oriented Computing: Building Multitier Dependability. *Computer*, 37(11):60–67, Nov. 2004a. Cited on 408

Candea G., Kawamoto S., Fujiki Y., Friedman G., and Fox A. Microreboot: A Technique for Cheap Recovery. In *6th Symposium on Operating System Design and Implementation*, pages 31–44, Berkeley, CA, Dec. 2004b. USENIX, USENIX. Cited on 408

Candea G., Kiciman E., Kawamoto S., and Fox A. Autonomous Recovery in Componentized Internet Applications. *Cluster Computing*, 9(2):175–190, Feb. 2006. Cited on 408

Cantin J., Lipasti M., and Smith J. The Complexity of Verifying Memory Coherence and Consistency. *IEEE Transactions on Parallel and Distributed Systems*, 16(7): 663–671, July 2005. Cited on 319

Cao L. and Ozsu T. Evaluation of Strong Consistency Web Caching Techniques. *World Wide Web*, 5(2):95–123, June 2002. Cited on 619

Cao P. and Liu C. Maintaining Strong Cache Consistency in the World Wide Web. *IEEE Transactions on Computers*, 47(4):445–457, Apr. 1998. Cited on 619

Caporuscio M., Carzaniga A., and Wolf A. L. Design and Evaluation of a Support Service for Mobile, Wireless Publish/Subscribe Applications. *IEEE Transactions on Software Engineering*, 29(12):1059–1071, Dec. 2003. Cited on 646

Cardellini V., Casalicchio E., Colajanni M., and Yu P. The State of the Art in Locally Distributed Web-Server Systems. *ACM Computing Surveys*, 34(2):263–311, June 2002. Cited on 605

Carriero N. and Gelernter D. The S/Net's Linda Kernel. *ACM Transactions on Computer Systems*, 32(2):110–129, May 1986. Cited on 657

Carzaniga A. and Wolf A. L. Forwarding in a Content-based Network. In *SIGCOMM*, pages 163–174, New York, NY, Aug. 2003. ACM, ACM Press. Cited on 650

Carzaniga A., Rutherford M. J., and Wolf A. L. A Routing Scheme for Content-Based Networking. In *23rd INFOCOM Conference*, Los Alamitos, CA., Mar. 2004. IEEE, IEEE Computer Society Press. Cited on 648

Castro M. and Liskov B. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Transactions on Computer Systems*, 20(4):398–461, Nov. 2002. Cited on 572, 574, 630

Castro M., Druschel P., Ganesh A., Rowstron A., and Wallach D. S. Secure Routing for Structured Peer-to-Peer Overlay Networks. In *5th Symposium on Operating System Design and Implementation*, pages 299–314, New York, NY, Dec. 2002a. USENIX, ACM Press. Cited on 583, 584

Castro M., Druschel P., Hu Y. C., and Rowstron A. Topology-aware Routing in Structured Peer-to-Peer Overlay Networks. Technical Report MSR-TR-2002-82, Microsoft Research, Cambridge, UK, June 2002b. Cited on 217

Castro M., Druschel P., Kermarrec A.-M., and Rowstron A. Scribe: A Large-Scale and Decentralized Application-Level Multicast Infrastructure. *IEEE Journal on Selected Areas in Communication*, 20(8):100–110, Oct. 2002c. Cited on 193, 194

Castro M., Rodrigues R., and Liskov B. BASE: Using Abstraction to Improve Fault

Tolerance. *ACM Transactions on Computer Systems*, 21(3):236–269, Aug. 2003. Cited on 574

Castro M., Costa M., and Rowstron A. Debunking Some Myths about Structured and Unstructured Overlays. In *2nd Symposium on Networked Systems Design and Implementation*, Berkeley, CA, Mar. 2005. USENIX, USENIX. Cited on 68

Cheriton D. and Mann T. Decentralizing a Global Naming Service for Improved Performance and Fault Tolerance. *ACM Transactions on Computer Systems*, 7(2): 147–183, May 1989. Cited on 230

Cheriton D. and Skeen D. Understanding the Limitations of Causally and Totally Ordered Communication. In *14th Symposium on Operating System Principles*, pages 44–57. ACM, Dec. 1993. Cited on 280

Chervenak A., Foster I., Kesselman C., Salisbury C., and Tuecke S. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. *J. Netw. Comp. App.*, 23(3):187–200, July 2000. Cited on 414

Chervenak A., Schuler R., Kesselman C., Koranda S., and Moe B. Wide Area Data Replication for Scientific Collaborations. In *6th Int'l Workshop on Grid Computing*, New York, NY, Nov. 2005. IEEE/ACM, ACM Press. Cited on 572

Cheswick W. and Bellovin S. *Firewalls and Internet Security*. Addison-Wesley, Reading, MA., 2nd edition, 2000. Cited on 456

Chow R. and Johnson T. *Distributed Operating Systems and Algorithms*. Addison-Wesley, Reading, MA., 1997. Cited on 398, 401

Chun B. and Spalink T. Slice Creation and Management. Technical Report PDN-03-013, PlanetLab Consortium, July 2003. Cited on 123

Ciancarini P., Tolksdorf R., Vitali F., and Knoche A. Coordinating Multiagent Applications on the WWW: A Reference Architecture. *IEEE Transactions on Software Engineering*, 24(5):362–375, May 1998. Cited on 658

Clark C., Fraser K., Hand S., Hansen J. G., Jul E., Limpach C., Pratt I., and Warfield A. Live Migration of Virtual Machines. In *2nd Symposium on Networked Systems Design and Implementation*, Berkeley, CA, May 2005. USENIX, USENIX. Cited on 134, 135

Clark D. The Design Philosophy of the DARPA Internet Protocols. In *SIGCOMM*, pages 106–114, New York, NY, Sept. 1989. ACM, ACM Press. Cited on 113

Clement L., Hately A., Riegen C.von , and Rogers T. Universal Description, Discovery and Integration (UDDI). Technical report, OASIS UDDI, 2004. Cited on 251

Cohen B. Incentives Build Robustness in Bittorrent. In *1st Workshop on Economics of Peer-to-Peer Systems*, June 2003. Cited on 72

Cohen D. On Holy Wars and a Plea for Peace. *Computer*, 14(10):48–54, Oct. 1981. Cited on 154

Cohen E. and Shenker S. Replication Strategies in Unstructured Peer-to-Peer Networks. In *SIGCOMM*, pages 177–190, New York, NY, Aug. 2002. ACM, ACM Press. Cited on 569

Comer D. *Internetworking with TCP/IP, Volume I: Principles, Protocols, and Architecture*. Prentice Hall, Upper Saddle River, N.J., 5th edition, 2006. Cited on 143

Conti M., Gregori E., and Lapenna W. Content Delivery Policies in ReplicatedWeb Services: Client-Side vs. Server-Side. *Cluster Computing*, 8:47–60, Jan. 2005. Cited

on 626

Coppersmith D. The Data Encryption Standard (DES) and its Strength Against Attacks. *IBM J. Research and Development*, 38(3):243–250, May 1994. Cited on 428

Cox L. and Noble B. Samsara: Honor Among Thieves in Peer-to-Peer Storage. In *19th Symposium on Operating System Principles*, pages 120–131, New York, NY, Oct. 2003. ACM, ACM Press. Cited on 584

Coyler A., Blair G., and Rashid A. Managing Complexity In Middleware. In *2nd AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software*, Lancaster, UK, Mar. 2003. Cited on 78

Crespo A. and Garcia-Moilina H. Semantic Overlay Networks for P2P Systems. Technical report, Stanford University, Department of Computer Science, 2003. Cited on 254

Cristian F. Probabilistic Clock Synchronization. *Distributed Computing*, 3:146–158, 1989. Cited on 268

Cristian F. Understanding Fault-Tolerant Distributed Systems. *Communications of the ACM*, 34(2):56–78, Feb. 1991. Cited on 356

Crowley C. *Operating Systems, A Design-Oriented Approach*. Irwin, Chicago, 1997. Cited on 224

Dabek F., Kaashoek M. F., Karger D., Morris R., and Stoica I. Wide-area Cooperative Storage with CFS. In *18th Symposium on Operating System Principles*, Baniff, Canada, Oct. 2001. ACM. Cited on 540

Dabek F., Cox R., Kaashoek F., and Morris R. Vivaldi: A Decentralized Network Coordinate System. In *SIGCOMM*, New York, NY, Aug. 2004a. ACM, ACM Press. Cited on 293

Dabek F., Li J., Sit E., Robertson J., Kaashoek M. F., and Morris R. Designing a dht for low latency and high throughput. In *1st Symposium on Networked Systems Design and Implementation*, pages 85–98, Berkeley, CA, Mar. 2004b. USENIX, USENIX. Cited on 218

Daemen J. and Rijmen V. Aes proposal: Rijndael. AES Algorithm Submission, Sept. 1999. `http://www.nist.gov/aes`. Cited on 428

Daigle L., Gulik D.van , Iannella R., and Faltstrom P. Uniform Resource Names (URN) Namespace Definition Mechanisms. RFC 3406, Oct. 2002. Cited on 614

Davie B., Charny A., Bennet J., Benson K., Boudec J. L., Courtney W., S.Davari , Firoiu V., and Stiliadis D. An Expedited Forwarding PHB (Per-Hop Behavior). RFC 3246, Mar. 2002. Cited on 187

Day J. and Zimmerman H. The OSI Reference Model. *Proceedings of the IEEE*, 71(12): 1334–1340, Dec. 1983. Cited on 139

Deering S. and Cheriton D. Multicast Routing in Datagram Internetworks and Extended LANs. *ACM Transactions on Computer Systems*, 8(2):85–110, May 1990. Cited on 210

Deering S., Estrin D., Farinacci D., Jacobson V., Liu C.-G., and Wei L. The PIM Architecture for Wide-Area Multicast Routing. *IEEE/ACM Transactions on Networking*, 4 (2):153–162, Apr. 1996. Cited on 210

Defago X., Shiper A., and Urban P. Total Order Broadcast and Multicast Algorithms: Taxonomy and Survey. *ACM Computing Surveys*, 36(4):372–421, Dec. 2004. Cited

on 377

Demers A., Greene D., Hauser C., Irish W., Larson J., Shenker S., Sturgis H., Swinehart D., and Terry D. Epidemic Algorithms for Replicated Database Maintenance. In *6th Symposium on Principles of Distributed Computing*, pages 1–12. ACM, Aug. 1987. Cited on 197, 199, 200

Demers A., Gehrke J., Hong M., Riedewald M., and White W. Towards Expressive Publish/Subscribe Systems. In *10th International Conference on Extended Database Technology*, Mar. 2006. Cited on 654

Deutsch P., Schoultz R., Faltstrom P., and Weider C. Architecture of the WHOIS++ Service. RFC 1835, Aug. 1995. Cited on 84

Diao Y., Hellerstein J., Parekh S., Griffith R., Kaiser G., and Phung D. A Control Theory Foundation for Self-Managing Computing Systems. *IEEE Journal on Selected Areas in Communication*, 23(12):2213–2222, Dec. 2005. Cited on 80

Dierks T. and Allen C. The Transport Layer Security Protocol. RFC 2246, Jan. 1996. Cited on 631

Diffie W. and Hellman M. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, Nov. 1976. Cited on 467

Dilley J., Maggs B., Parikh J., Prokop H., Sitaraman R., and Weihl B. Globally Distributed Content Delivery. *IEEE Internet Computing*, 6(5):50–58, Sept. 2002. Cited on 624

Diot C., Levine B., Lyles B., Kassem H., and Balensiefen D. Deployment Issues for the IP Multicast Service and Architecture. *IEEE Network*, 14(1):78–88, Jan. 2000. Cited on 193

Doorn J. H. and Rivero L. C., editors. *Database Integrity: Challenges and Solutions*. Idea Group, Hershey, PA, 2002. Cited on 418

Douceur J. R. The Sybil Attack. In *1st International Workshop on Peer-to-Peer Systems*, volume 2429 of *Lecture Notes in Computer Science*, pages 251–260, Berlin, Mar. 2002. Springer-Verlag. Cited on 583

Dubois M., Scheurich C., and Briggs F. Synchronization, Coherence, and Event Ordering in Multiprocessors. *Computer*, 21(2):9–21, Feb. 1988. Cited on 313

Dunagan J., Harvey N. J. A., Jones M. B., Kostic D., Theimer M., and Wolman A. FUSE: Lightweight Guaranteed Distributed Failure Notification. In *6th Symposium on Operating System Design and Implementation*, Berkeley, CA, Dec. 2004. USENIX, USENIX. Cited on 369

Duvvuri V., Shenoy P., and Tewari R. Adaptive Leases: A Strong Consistency Mechanism for the World Wide Web. *IEEE Transactions on Knowledge and Data Engineering*, 15(5):1266–1276, Sept. 2003. Cited on 337

Eddon G. and Eddon H. *Inside Distributed COM*. Microsoft Press, Redmond, WA, 1998. Cited on 159

Eisler M. LIPKEY - A Low Infrastructure Public Key Mechanism Using SPKM. RFC 2847, June 2000. Cited on 578

Eisler M., Chiu A., and Ling L. RPCSEC_GSS Protocol Specification. RFC 2203, Sept. 1997. Cited on 577

El-Sayed A., Roca V., and Mathy L. A Survey of Proposals for an Alternative Group Communication Service. *IEEE Network*, 17(1):46–51, Jan. 2003. Cited on 193

Elnozahy E., Alvisi L., Wang Y.-M., and Johnson D. A Survey of Rollback-Recovery Protocols in Message-Passing Systems. *ACM Computing Surveys*, 34(3):375–408, Sept. 2002. Cited on 401, 408

Elnozahy E. N. and Plank J. S. Checkpointing for Peta-Scale Systems: A Look into the Future of Practical Rollback-Recovery. *IEEE Transactions on Dependable and Secure Computing*, 1(2):97–108, Apr. 2004. Cited on 404

Elson J., Girod L., and Estrin D. Fine-Grained Network Time Synchronization using Reference Broadcasts. In *5th Symposium on Operating System Design and Implementation*, pages 147–163, New York, NY, Dec. 2002. USENIX, ACM Press. Cited on 271, 272

Eugster P., Felber P., Guerraoui R., and Kermarrec A.-M. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114–131, June 2003. Cited on 53, 637

Eugster P., Guerraoui R., Kermarrec A.-M., and Massoulié L. Epidemic Information Dissemination in Distributed Systems. *Computer*, 37(5):60–67, May 2004. Cited on 197

Farmer W. M., Guttman J. D., and Swarup V. Security for Mobile Agents: Issues and Requirements. In *19th National Information Systems Security Conference*, pages 591–597, Baltimore, MD, Oct. 1996. Cited on 458

Felber P. and Narasimhan P. Experiences, Strategies, and Challenges in Building Fault-Tolerant CORBA Systems. *Computer*, 53(5):497–511, May 2004. Cited on 520

Ferguson N. and Schneier B. *Practical Cryptography*. John Wiley, New York, 2003. Cited on 426, 436

Fielding R., Gettys J., Mogul J., Frystyk H., Masinter L., Leach P., and Berners-Lee T. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, June 1999. Cited on 144, 606

Filman R. E., Elrad T., Clarke S., and Aksit M., editors. *Aspect-Oriented Software Development*. Addison-Wesley, Reading, MA., 2005. Cited on 77

Fischer M., Lynch N., and Patterson M. Impossibility of Distributed Consensus with one Faulty Processor. *Journal of the ACM*, 32(2):374–382, Apr. 1985. Cited on 367

Floyd S., Jacobson V., McCanne S., Liu C.-G., and Zhang L. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. *IEEE/ACM Transactions on Networking*, 5(6):784–803, Dec. 1997. Cited on 379, 380

Foster I. and Kesselman C. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufman, San Mateo, CA., 2nd edition, 2003. Cited on 414

Foster I., Kesselman C., Tsudik G., and Tuecke S. A Security Architecture for Computational Grids. In *5th Conference on Computer and Communications Security*, pages 83–92, San Francisco, CA, Nov. 1998. ACM. Cited on 414, 416, 418

Foster I., Kesselman C., and Tuecke S. The Anatomy of the Grid, Enabling Scalable Virtual Organizations. *Journal of Supercomputer Applications*, 15(3):200–222, Fall 2001. Cited on

Foster I. and others . The Open Grid Services Architecture, Version 1.5. GGF Informational Document GFD-I.080, June 2006. Cited on

Fowler R. *Decentralized Object Finding Using Forwarding Addresses*. Ph.D., University of Washington, Seattle, 1985. Cited on 210

Franklin M. J., Carey M. J., and Livny M. Transactional Client-Server Cache Consis-

tency: Alternatives and Performance. *ACM Transactions on Database Systems*, 22(3): 315–363, Sept. 1997. Cited on 347, 348

Freeman E., Hupfer S., and Arnold K. *JavaSpaces, Principles, Patterns and Practice*. Addison-Wesley, Reading, MA., 1999. Cited on 639, 640

Freund R. Web Services Coordination, Version 1.0, Feb. 2005. Cited on 598

Friedman R. and Kama A. Transparent Fault-Tolerant Java Virtual Machine. In *22nd Symposium on Reliable Distributed Systems*, pages 319–328, IEEE Computer Society Press, Oct. 2003. IEEE, IEEE Computer Society Press. Cited on 520, 521

Fuggetta A., Picco G. P., and Vigna G. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, May 1998. Cited on 128, 131

Gamma E., Helm R., Johnson R., and Vlissides J. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA., 1994. Cited on 455, 484

Garbacki P., Epema D., and Steen M.van . A Two-Level Semantic Caching Scheme for Super-Peer Networks. In *10th Web Caching Workshop*. IEEE, Sept. 2005. Cited on 71

Garcia-Molina H. Elections in a Distributed Computing System. *IEEE Transactions on Computers*, 31(1):48–59, Jan. 1982. Cited on 294

Garman J. *Kerberos: The Definitive Guide*. O'Reilly & Associates, Sebastopol, CA., 2003. Cited on 448

Gelernter D. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985. Cited on 637

Gelernter D. and Carriero N. Coordination Languages and their Significance. *Communications of the ACM*, 35(2):96–107, Feb. 1992. Cited on 636

Ghemawat S., Gobioff H., and Leung S.-T. The Google File System. In *19th Symposium on Operating System Principles*, pages 29–43, New York, NY, Oct. 2003. ACM, ACM Press. Cited on 537

Gifford D. Weighted Voting for Replicated Data. In *7th Symposium on Operating System Principles*, pages 150–162. ACM, Dec. 1979. Cited on 345

GigaSpaces . *GigaSpaces Cache 5.0 Documentation*. New York, NY, 2005. Cited on 658

Gil T. M. and Poletto M. MULTOPS: a Data-Structure for Bandwidth Attack Detection. In *10th USENIX Security Symposium*, pages 23–38, Berkeley, CA, Aug. 2001. USENIX, USENIX. Cited on 466

Gladney H. Access Control for Large Collections. *ACM Transactions on Information Systems*, 15(2):154–194, Apr. 1997. Cited on 455

Goland Y., Whitehead E., Faizi A., Carter S., and Jensen D. HTTP Extensions for Distributed Authoring – WEBDAV. RFC 2518, Feb. 1999. Cited on 616

Gollmann D. *Computer Security*. John Wiley, New York, 2nd edition, 2006. Cited on 418

Gong L. and Schemers R. Implementing Protection Domains in the Java Development Kit 1.2. In *Symposium on Network and Distributed System Security*, pages 125–134, San Diego, CA, Mar. 1998. Internet Society. Cited on 464

Gopalakrishnan V., Silaghi B., Bhattacharjee B., and Keleher P. Adaptive Replication in Peer-to-Peer Systems. In *24th International Conference on Distributed Computing Systems*, pages 360–369, Los Alamitos, CA., Mar. 2004. IEEE, IEEE Computer Society Press. Cited on 570

Gray C. and Cheriton D. Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency. In *12th Symposium on Operating System Principles*, pages 202–210, New York, NY, Dec. 1989. ACM, ACM Press. Cited on 337

Gray J. Notes on Database Operating Systems. In Bayer R., Graham R., and Seegmuller G., editors, *Operating Systems: An Advanced Course*, volume 60 of *Lecture Notes in Computer Science*, pages 393–481. Springer-Verlag, Berlin, 1978. Cited on 390

Gray J. and Reuter A. *Transaction Processing: Concepts and Techniques*. Morgan Kaufman, San Mateo, CA., 1993. Cited on

Gray J., Helland P., O'Neil P., and Sashna D. The Dangers of Replication and a Solution. In *SIGMOD International Conference on Management Of Data*, pages 173–182, Montreal, June 1996. ACM. Cited on 306

Grimm R., Davis J., Lemar E., Macbeth A., Swanson S., Anderson T., Bershad B., Borriello G., Gribble S., and Wetherall D. System Support for Pervasive Applications. *ACM Transactions on Computer Systems*, 22(4):421–486, Nov. 2004. Cited on

Gropp W., Huss-Lederman S., Lumsdaine A., Lusk E., Nitzberg B., Saphir W., and Snir M. *MPI: The Complete Reference – The MPI-2 Extensions*. MIT Press, Cambridge, MA., 1998. Cited on 169

Gropp W., Lusk E., and Skjellum A. *Using MPI-2, Portable Parallel Programming with the Message-Passing Interface*. MIT Press, Cambridge, MA., 2nd edition, 1999. Cited on 169

Grosskurth A. and Godfrey M. W. A Reference Architecture for Web Browsers. In *21st Int'l Conf. Softw. Mainten.*, pages 661–664, Los Alamitos, CA., Sept. 2005. IEEE, IEEE Computer Society Press. Cited on 599

Gudgin M., Hadley M., Mendelsohn N., Moreau J.-J., and Nielsen H. F. SOAP Version 1.2. W3C Recommendation, June 2003. Cited on 611, 613

Guerraoui R. and Rodrigues L. *Introduction to Reliable Distributed Programming*. Springer-Verlag, Berlin, 2006. Cited on 259

Guerraoui R. and Schiper A. Software-Based Replication for Fault Tolerance. *Computer*, 30(4):68–74, Apr. 1997. Cited on 360

Guichard J., Faucheur F. L., and Vasseur J.-P. *Definitive MPLS Network Designs*. Cisco Press, Indianapolis, IN, 2005. Cited on 621

Gulbrandsen A., Vixie P., and Esibov L. A dns rr for specifying the location of services (dns srv). RFC 2782, Feb. 2000. Cited on 239

Gupta A., Sahin O. D., Agrawal D., and Abbadi A. E. Meghdoot: Content-Based Publish/Subscribe over P2P Networks. In *Middleware 2004*, volume 3231 of *Lecture Notes in Computer Science*, pages 254–273, Berlin, Oct. 2004. ACM/IFIP/USENIX, Springer-Verlag. Cited on 646

Gusella R. and Zatti S. The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3BSD. *IEEE Transactions on Software Engineering*, 15 (7):847–853, July 1989. Cited on 270

Hadzilacos V. and Toueg S. Fault-Tolerant Broadcasts and Related Problems. In Mullender S., editor, *Distributed Systems*, pages 97–145. Addison-Wesley, Wokingham, 2nd edition, 1993. Cited on 356, 387

Halsall F. *Multimedia Communications: Applications, Networks, Protocols and Standards*. Addison-Wesley, Reading, MA., 2001. Cited on 184, 186

Handurukande S., Kermarrec A.-M., Fessant F. L., and Massoulié L. Exploiting Semantic Clustering in the eDonkey P2P network. In *11th SIGOPS European Workshop*, New York, NY, Sept. 2004. ACM, ACM Press. Cited on 255

Helder D. A. and Jamin S. End-Host Multicast Communication Using Switch-Trees Protocols. In *2nd International Symposium on Cluster Computing and the Grid*, pages 419–424, Los Alamitos, CA., May 2002. IEEE, IEEE Computer Society Press. Cited on 196

Hellerstein J. L., Diao Y., Parekh S., and Tilbury D. M. *Feedback Control of Computing Systems*. John Wiley, New York, 2004. Cited on 80

Henning M. A New Approach to Object-Oriented Middleware. *IEEE Internet Computing*, 8(1):66–75, Jan. 2004. Cited on 493

Henning M. and Spruiell M. *Distributed Programming with Ice*. ZeroC Inc., Brisbane, Australia, May 2005. Cited on 493, 510

Hochstetler S. and Beringer B. Linux Clustering with CSM and GPFS. Technical Report SG24-6601-02, International Technical Support Organization, IBM, Austin, TX, Jan. 2004. Cited on 120

Hohpe G. and Woolf B. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, Reading, MA., 2004. Cited on 177

Horowitz M. and Lunt S. FTP Security Extensions. RFC 2228, Oct. 1997. Cited on 144

Howes T. The String Representation of LDAP Search Filters. RFC 2254, Dec. 1997. Cited on 250

Chu Y.hua , Rao S. G., Seshan S., and Zhang H. A Case for End System Multicast. *IEEE Journal on Selected Areas in Communication*, 20(8):1456–1471, Oct. 2002. Cited on 195

Huffaker B., Fomenkov M., Plummer D. J., Moore D., and Claffy K. Distance Metrics in the Internet. In *International Telecommunications Symposium*, Los Alamitos, CA., Sept. 2002. IEEE, IEEE Computer Society Press. Cited on 621

Hunt G., Nahum E., and Tracey J. Enabling Content-Based Load Distribution for Scalable Services. Technical report, IBM T.J. Watson Research Center, May 1997. Cited on 116

Hutto P. and Ahamad M. Slow Memory: Weakening Consistency to Enhance Concurrency in Distributed Shared Memories. In *10th International Conference on Distributed Computing Systems*, pages 302–311, Paris, France, May 1990. IEEE. Cited on 315

IBM. *WebSphere MQ Publish/Subscribe User's Guide*. IBM Inc., May 2005a. Cited on 639

IBM. *WebSphere MQ Intercommunication*. IBM Inc., May 2005b. Cited on 177

IBM. *WebSphere MQ System Administration*. IBM Inc., May 2005c. Cited on 177

IBM. *WebSphere MQ Application Programming Guide*. IBM Inc., May 2005d. Cited on 178

ISO . Open Distributed Processing Reference Model. International Standard ISO/IEC IS 10746, 1995. Cited on

Jaeger T., Prakash A., Liedtke J., and Islam N. Flexible Control of Downloaded Executable Content. *ACM Transactions on Information and System Security*, 2(2): 177–228, May 1999. Cited on 465

Jalote P. *Fault Tolerance in Distributed Systems*. Prentice Hall, Englewood Cliffs, N.J., 1994. Cited on 346, 354

Janic M. *Multicast in Network and Application Layer*. Ph.d., Delft University of Technology, The Netherlands, Oct. 2005. Cited on 193

Janiga M. J., Dibner G., and Governali F. J. Internet Infrastructure: Content Delivery. Goldman Sachs Global Equity Research, Apr. 2001. Cited on 622

Jelasity M. and Kermarrec A.-M. Ordered Slicing of Very Large-Scale Overlay Networks. In *6th International Conference on Peer-to-Peer Computing*, pages 117–124, Los Alamitos, CA., Sept. 2006. IEEE Computer Society Press. Cited on 68, 69

Jelasity M., Guerraoui R., Kermarrec A.-M., and Steen M.van . The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-Based Implementations. In *Middleware 2004*, volume 3231 of *Lecture Notes in Computer Science*, pages 79–98, Berlin, Oct. 2004. ACM/IFIP/USENIX, Springer-Verlag. Cited on 66

Jelasity M., Montresor A., and Babaoglu O. Gossip-based Aggregation in Large Dynamic Networks. *ACM Transactions on Computer Systems*, 23(3):219–252, Aug. 2005a. Cited on 201

Jelasity M., Voulgaris S., Guerraoui R., Kermarrec A.-M., and Steen M.van . Gossip-based Peer Sampling. Technical report, Vrije Universiteit, Department of Computer Science, Sept. 2005b. Cited on 66, 67, 68, 198, 255

Jin J. and Nahrstedt K. QoS Specification Languages for Distributed Multimedia Applications: A Survey and Taxonomy. *IEEE Multimedia*, 11(3):74–87, July 2004. Cited on 186

Jing J., Helal A., and Elmagarmid A. Client-Server Computing in Mobile Environments. *ACM Computing Surveys*, 31(2):117–157, June 1999. Cited on 60

Johnson B. An Introduction to the Design and Analysis of Fault-Tolerant Systems. In Pradhan D., editor, *Fault-Tolerant Computer System Design*, pages 1–87. Prentice Hall, Upper Saddle River, N.J., 1995. Cited on 358

Johnson D., Perkins C., and Arkko J. Mobility Support for IPv6. RFC 3775, June 2004. Cited on 213

Joseph J., Ernest M., and Fellenstein C. Evolution of grid computing architecture and grid adoption models. *IBM Systems Journal*, 43(4):624–645, Apr. 2004. Cited on

Jul E., Levy H., Hutchinson N., and Black A. Fine-Grained Mobility in the Emerald System. *ACM Transactions on Computer Systems*, 6(1):109–133, Feb. 1988. Cited on 212

Jung J., Sit E., Balakrishnan H., and Morris R. DNS Performance and the Effectiveness of Caching. *IEEE/ACM Transactions on Networking*, 10(5):589 – 603, Oct. 2002. Cited on 245

Kahn D. *The Codebreakers*. Macmillan, New York, 1967. Cited on 426

Kaminsky M., Savvides G., MaziÃĺres D., and Kaashoek M. F. Decentralized User Authentication in a Global File System. In *19th Symposium on Operating System Principles*, pages 60–73, New York, NY, Oct. 2003. ACM, ACM Press. Cited on 580,

582

Kantarcioglu M. and Clifton C. Security Issues in Querying Encrypted Data. In *19th Conf. Data & Appl. Security*, volume 3654 of *Lecture Notes in Computer Science*, pages 325–337, Berlin, Aug. 2005. IFIP, Springer-Verlag. Cited on 667

Karnik N. and Tripathi A. Security in the Ajanta Mobile Agent System. *Software – Practice and Experience*, 31(4):301–329, Apr. 2001. Cited on 458

Kasera S., Kurose J., and Towsley D. Scalable Reliable Multicast Using Multiple Multicast Groups. In *International Conference on Measurements and Modeling of Computer Systems*, pages 64–74, Seattle, WA, June 1997. ACM. Cited on 380

Katz E., Butler M., and McGrath R. A Scalable HTTP Server: The NCSA Prototype. *Computer Networks and ISDN Systems*, 27(2):155–164, Sept. 1994. Cited on 97

Kaufman C., Perlman R., and Speciner M. *Network Security: Private Communication in a Public World*. Prentice Hall, Englewood Cliffs, N.J., 2nd edition, 2003. Cited on 435, 443

Kent S. Internet Privacy Enhanced Mail. *Communications of the ACM*, 36(8):48–60, Aug. 1993. Cited on 470

Kephart J. O. and Chess D. M. The Vision of Autonomic Computing. *Computer*, 36 (1):41–50, Jan. 2003. Cited on 79

Khoshafian S. and Buckiewicz M. *Introduction to Groupware, Workflow, and Workgroup Computing*. John Wiley, New York, 1995. Cited on 177

Khurana H. and Koleva R. Scalable Security and Accounting Services for Content-Based Publish Subscribe Systems. *International Journal of E-Business Research*, 2, 2006. Cited on 666, 667, 668

Kim S., Pan K., Sinderson E., and Whitehead J. Architecture and Data Model of a WebDAV-based Collaborative System. In *Collaborative Techn. Symp.*, pages 48–55, Jan. 2004. Cited on 616

Kistler J. and Satyanaryanan M. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1):3–25, Feb. 1992. Cited on 544, 561

Kleiman S. Vnodes: an Architecture for Multiple File System Types in UNIX. In *Summer Technical Conference*, pages 238–247, Atlanta, GA, June 1986. USENIX. Cited on 532

Kohl J., Neuman B., and T'so T. The Evolution of the Kerberos Authentication System. In Brazier F. and Johansen D., editors, *Distributed Open Systems*, pages 78–94. IEEE Computer Society Press, Los Alamitos, CA., 1994. Cited on 448

Kon F., Costa F., Campbell R., and Blair G. The Case for Reflective Middleware. *Communications of the ACM*, 45(6):33–38, June 2002. Cited on 77

Kopetz H. and Verissimo P. Real Time and Dependability Concepts. In Mullender S., editor, *Distributed Systems*, pages 411–446. Addison-Wesley, Wokingham, 2nd edition, 1993. Cited on 354

Kostoulas M. G., Matsa M., Mendelsohn N., Perkins E., Heifets A., and Mercaldi M. XML Screamer: An Integrated Approach to High Performance XML Parsing, Validation and Deserialization. In *15th International World Wide Web Conference*, New York, NY, May 2006. ACM, ACM Press. Cited on 613

Kumar P. and Satyanarayanan M. Flexible and Safe Resolution of File Conflicts. In *Winter Technical Conference*, pages 95–106, New Orleans, LA, Jan. 1995. USENIX.

Cited on 568

Lai A. and Nieh J. Limits of Wide-Area Thin-Client Computing. In *International Conference on Measurements and Modeling of Computer Systems*, pages 228–239, New York, NY, June 2002. ACM, ACM Press. Cited on 105

LaMacchia B. and Odlyzko A. Computation of Discrete Logarithms in Prime Fields. *Designs, Codes, and Cryptography*, 1(1):47–62, May 1991. Cited on 577

Lamport L. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978. Cited on 273

Lamport L. How to Make a Multiprocessor Computer that Correctly Executes Multiprocessor Programs. *IEEE Transactions on Computers*, C-29(9):690–691, Sept. 1979. Cited on 312

Lamport L., Shostak R., and Paese M. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982. Cited on 358, 365, 367

Lampson B., Abadi M., Burrows M., and Wobber E. Authentication in Distributed Systems: Theory and Practice. *ACM Transactions on Computer Systems*, 10(4): 265–310, Nov. 1992. Cited on 432

Laprie J.-C. Dependability – Its Attributes, Impairments and Means. In Randell B., Laprie J.-C., Kopetz H., and Littlewood B., editors, *Predictably Dependable Computing Systems*, pages 3–24. Springer-Verlag, Berlin, 1995. Cited on 412

Laurie B. and Laurie P. *Apache: The Definitive Guide*. O'Reilly & Associates, Sebastopol, CA., 3rd edition, 2002. Cited on 603

Leff A. and Rayfield J. T. Alternative Edge-server Architectures for Enterprise JavaBeans Applications. In *Middleware 2004*, volume 3231 of *Lecture Notes in Computer Science*, pages 195–211, Berlin, Oct. 2004. ACM/IFIP/USENIX, Springer-Verlag. Cited on 72

Leighton F. and Lewin D. Global Hosting System. United States Patent, Number 6,108,703, Aug. 2000. Cited on 624

Levien R., editor. *Signposts in Cyberspace: The Domain Name System and Internet Navigation*. National Academic Research Council, Washington, DC, 2005. Cited on 238

Levine B. and Garcia-Luna-Aceves J. A Comparison of Reliable Multicast Protocols. *ACM Multimedia Systems Journal*, 6(5):334–348, 1998. Cited on 379

Lewis B. and Berg D. J. *Multithreaded Programming with Pthreads*. Prentice Hall, Englewood Cliffs, N.J., 2nd edition, 1998. Cited on 90

Li G. and Jacobsen H.-A. Composite Subscriptions in Content-Based Publish/Subscribe Systems. In *Middleware 2005*, volume 3790 of *Lecture Notes in Computer Science*, pages 249–269, Berlin, Nov. 2005. ACM/IFIP/USENIX, Springer-Verlag. Cited on 651

Li J., Lu C., and Shi W. An Efficient Scheme for Preserving Confidentiality in Content-Based Publish-Subscribe Systems. Technical Report GIT-CC-04-01, Georgia Institute of Technology, College of Computing, 2004a. Cited on 667

Li N., Mitchell J. C., and Tong D. Securing Java RMI-based Distributed Applications. In *20th Annual Computer Security Applications Conference*. ACSA, Dec. 2004b. Cited on 527

Lilja D. Cache Coherence in Large-Scale Shared-Memory Multiprocessors: Issues and Comparisons. *ACM Computing Surveys*, 25(3):303–338, Sept. 1993. Cited on 347

Lin M.-J. and Marzullo K. Directional Gossip: Gossip in a Wide-Area Network. In *Proceedings 3rd European Dependable Computing Conf.*, volume 1667 of *Lecture Notes in Computer Science*, pages 364–379. Springer-Verlag, Berlin, Sept. 1999. Cited on 199

Lin S.-D., Lian Q., Chen M., , and Zhang Z. A Practical Distributed Mutual Exclusion Protocol in Dynamic Peer-to-Peer Systems. In *3rd International Workshop on Peer-to-Peer Systems*, volume 3279 of *Lecture Notes in Computer Science*, pages 11–21, Berlin, Feb. 2004. Springer-Verlag. Cited on 283, 284

Ling B. C., Kiciman E., and Fox A. Session State: Beyond Soft State. In *1st Symposium on Networked Systems Design and Implementation*, pages 295–308, Berkeley, CA, Mar. 2004. USENIX, USENIX. Cited on 113

Linn J. Generic Security Service Application Program Interface, version 2. RFC 2078, Jan. 1997. Cited on 578

Liu C. and Albitz P. *DNS and BIND*. O'Reilly & Associates, Sebastopol, CA., 5th edition, 2006. Cited on 237

Liu C.-G., Estrin D., Shenker S., and Zhang L. Local Error Recovery in SRM: Comparison of Two Approaches. *IEEE/ACM Transactions on Networking*, 6(6): 686–699, Dec. 1998. Cited on 380

Liu H. and Jacobsen H.-A. Modeling Uncertainties in Publish/Subscribe Systems. In *20th International Conference on Data Engineering*, pages 510–522, Los Alamitos, CA., Mar. 2004. IEEE, IEEE Computer Society Press. Cited on 654

Lo V., Zhou D., Liu Y., GauthierDickey C., and Li J. Scalable Supernode Selection in Peer-to-Peer Overlay Networks. In *2nd Hot Topics in Peer-to-Peer Systems*, pages 18–27, Los Alamitos, CA., July 2005. IEEE Computer Society Press. Cited on 299

Lua E., Crowcroft J., Pias M., Sharma R., and Lim S. A Survey and Comparison of Peer-to-Peer Overlay Network Schemes. *IEEE Communications Surveys & Tutorials*, 7(2):22–73, Apr. 2005. Cited on 63

Lui J., Misra V., and Rubenstein D. On the Robustness of Soft State Protocols. In *12th International Conference on Network Protocols*, pages 50–60, Los Alamitos, CA., Oct. 2004. IEEE, IEEE Computer Society Press. Cited on 113

Luotonen A. and Altis K. World-Wide Web Proxies. *Computer Networks and ISDN Systems*, 27(2):1845–1855, 1994. Cited on 600

Lynch N. *Distributed Algorithms*. Morgan Kaufman, San Mateo, CA., 1996. Cited on 259, 294

Maassen J., Kielmann T., and Bal H. E. Parallel Application Experience with Replicated Method Invocation. *Concurrency & Computation: Practice and Experience*, 13 (8-9):681–712, 2001. Cited on 516

Macgregor R., Durbin D., Owlett J., and Yeomans A. *Java Network Security*. Prentice Hall, Upper Saddle River, N.J., 1998. Cited on 460

Madden S. R., Franklin M. J., Hellerstein J. M., and Hong W. TinyDB: An Acquisitional Query Processing System for Sensor Networks. *ACM Transactions on Database Systems*, 30(1):122–173, 2005. Cited on

Makpangou M., Gourhant Y., Narzul J.-P.le , and Shapiro M. Fragmented Objects for Distributed Abstractions. In Casavant T. and Singhal M., editors, *Readings in Distributed Computing Systems*, pages 170–186. IEEE Computer Society Press, Los Alamitos, CA., 1994. Cited on 487

Malkhi D. and Reiter M. Secure Execution of Java Applets using a Remote Playground. *IEEE Transactions on Software Engineering*, 26(12):1197–1209, Dec. 2000. Cited on 462

Mamei M. and Zambonelli F. Programming Pervasive and Mobile Computing Applications with the TOTA Middleware. In *2nd International Conference on Pervasive Computing and Communications (PerCom)*, pages 263–273, Los Alamitos, CA., Mar. 2004. IEEE, IEEE Computer Society Press. Cited on 648

Manola F. and Miller E. RDF Primer. W3C Recommendation, Feb. 2004. Cited on 247

Mascolo C., Capra L., and Emmerich W. Principles of Mobile Computing Middleware. In Mahmoud Q. H., editor, *Middleware for Communications*, chapter 12. John Wiley, New York, 2004. Cited on

Masinter L. The Data URL Scheme. RFC 2397, Aug. 1998. Cited on 615

Mazieres D., Kaminsky M., Kaashoek M., and Witchel E. Separating Key Management from File System Security. In *17th Symposium on Operating System Principles*, pages 124–139, Kiawah Island, SC, Dec. 1999. ACM. Cited on 525, 580

Mazouni K., Garbinato B., and Guerraoui R. Building Reliable Client-Server Software Using Actively Replicated Objects. In Graham I., Magnusson B., Meyer B., and Nerson J.-M., editors, *Technology of Object Oriented Languages and Systems*, pages 37–53. Prentice Hall, Englewood Cliffs, N.J., 1995. Cited on 516

McKinley P., Sadjadi S., Kasten E., and Cheng B. Composing Adaptive Software. *Computer*, 37(7):56–64, Jan. 2004. Cited on 77

Mehta N., Medvidovic N., and Phadke S. Towards A Taxonomy Of Software Connectors. In *22nd International Conference on Software Engineering*, pages 178–187, New York, NY, June 2000. ACM, ACM Press. Cited on 52

Menezes A. J., Oorschot P. C.van , and Vanstone S. A. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, 3rd edition, 1996. Cited on 426, 469, 470

Merideth M. G., Iyengar A., Mikalsen T., Tai S., Rouvellou I., and Narasimhan P. Thema: Byzantine-Fault-Tolerant Middleware for Web-Service Applications. In *24th Symposium on Reliable Distributed Systems*, pages 131–142, Los Alamitos, CA., Oct. 2005. IEEE, IEEE Computer Society Press. Cited on 630, 631

Meyer B. *Object-Oriented Software Construction*. Prentice Hall, Englewood Cliffs, N.J., 2nd edition, 1997. Cited on 483

Miller B. N., Konstan J. A., and Riedl J. PocketLens: Toward a Personal Recommender System. *ACM Transactions on Information Systems*, 22(3):437–476, July 2004. Cited on

Mills D. Network Time Protocol (version 3): Specification, Implementation, and Analysis. RFC 1305, July 1992. Cited on 270

Mills D. L. *Computer Network Time Synchronization: The Network Time Protocol*. CRC Press, Boca Raton, FL, 2006. Cited on 270

Milojicic D., Douglis F., Paindaveine Y., Wheeler R., and Zhou S. Process Migration.

ACM Computing Surveys, 32(3):241–299, Sept. 2000. Cited on 126

Min S. L. and Baer J.-L. Design and Analysis of a Scalable Cache Coherence Scheme Based on Clocks and Timestamps. *IEEE Transactions on Parallel and Distributed Systems*, 3(1):25–44, Jan. 1992. Cited on 347

Mirkovic J. and Reiher P. A Taxonomy of DDoS Attack and DDoS Defense Mechanisms. *ACM Computer Communications Review*, 34(2):39–53, Apr. 2004. Cited on 466

Mirkovic J., Dietrich S., and Reiher D. D.andPeter . *Internet Denial of Service: Attack and Defense Mechanisms*. Prentice Hall, Englewood Cliffs, N.J., 2005. Cited on 466

Mockapetris P. Domain Names - Concepts and Facilities. RFC 1034, Nov. 1987. Cited on 231, 237

Monson-Haefel R., Burke B., and Labourey S. *Enterprise Java Beans*. O'Reilly & Associates, Sebastopol, CA., 4th edition, 2004. Cited on 485

Moser L., Melliar-Smith P., Agarwal D., Budhia R., and Lingley-Papadopoulos C. Totem: A Fault-Tolerant Multicast Group Communication System. *Communications of the ACM*, 39(4):54–63, Apr. 1996. Cited on 519

Moser L., Mellior-Smith P., and Narasimhan P. Consistent Object Replication in the Eternal System. *Theory and Practice of Object Systems*, 4(2):81–92, 1998. Cited on 519

Mullender S. and Tanenbaum A. Immediate Files. *Software – Practice and Experience*, 14(3):365–368, 1984. Cited on 615

Muntz D. and Honeyman P. Multi-level Caching in Distributed File Systems. In *Winter Technical Conference*, pages 305–313, San Francisco, CA, Jan. 1992. USENIX. Cited on 333

Murphy A., Picco G., and Roman G.-C. Lime: A Middleware for Physical and Logical Mobility. In *21st International Conference on Distributed Computing Systems*, pages 524–533, Los Alamitos, CA., Apr. 2001. IEEE, IEEE Computer Society Press. Cited on 646

Muthitacharoen A., Morris R., Gil T., and Chen B. Ivy: A Read/Write Peer-to-Peer File System. In *5th Symposium on Operating System Design and Implementation*, pages 31–44, New York, NY, Dec. 2002. ACM, ACM Press. Cited on 539

Napper J., Alvisi L., and Vin H. M. A Fault-Tolerant Java Virtual Machine. In *International Conference on Dependable Systems and Networks*, pages 425–434, Los Alamitos, CA., June 2003. IEEE Computer Society Press. Cited on 520

Narasimhan P., Moser L., and Melliar-Smith P. The Eternal System. In Urban J. and Dasgupta P., editors, *Encyclopedia of Distributed Computing*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000. Cited on 519

Nayate A., Dahlin M., and Iyengar A. Transparent Information Dissemination. In *Middleware 2004*, volume 3231 of *Lecture Notes in Computer Science*, pages 212–231, Berlin, Oct. 2004. ACM/IFIP/USENIX, Springer-Verlag. Cited on 72

Needham R. and Schroeder M. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, Dec. 1978. Cited on 437

Nelson B. *Remote Procedure Call*. Ph.D., Carnegie-Mellon University, 1981. Cited on 375

Neuman B. Proxy-Based Authorization and Accounting for Distributed Systems. In *13th International Conference on Distributed Computing Systems*, pages 283–291, Pittsburgh, May 1993. IEEE. Cited on 476

Neuman B. Scale in Distributed Systems. In Casavant T. and Singhal M., editors, *Readings in Distributed Computing Systems*, pages 463–489. IEEE Computer Society Press, Los Alamitos, CA., 1994. Cited on

Neuman C., Yu T., Hartman S., and Raeburn K. The Kerberos Network Authentication Service. RFC 4120, July 2005. Cited on 448

Neumann P. Architectures and Formal Representations for Secure Systems. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, Oct. 1995. Cited on 422

Ng E. and Zhang H. Predicting Internet Network Distance with Coordinates-Based Approaches. In *21st INFOCOM Conference*, Los Alamitos, CA., June 2002. IEEE, IEEE Computer Society Press. Cited on 292

Niemela E. and Latvakoski J. Survey of Requirements and Solutions for Ubiquitous Software. In *3rd International Conference on Mobile and Ubiquitous Multimedia*, pages 71–78, 2004. Cited on

Noble B., Fleis B., and Kim M. A Case for Fluid Replication. In *NetStore'99*, Seattle, WA, Oct. 1999. Cited on 334

Obraczka K. Multicast Transport Protocols: A Survey and Taxonomy. *IEEE Communications Magazine*, 36(1):94–102, Jan. 1998. Cited on 193

OMG . The Common Object Request Broker: Core Specification, revision 3.0.3. OMG Document formal/04-03-12, Object Management Group, Framingham, MA, Mar. 2004a. Cited on 74, 492, 504, 518

OMG . UML 2.0 Superstructure Specification. OMG Document ptc/04-10-02, Object Management Group, Framingham, MA, Oct. 2004b. Cited on 52

Oppenheimer D., Albrecht J., Patterson D., and Vahdat A. Design and Implementation Tradeoffs for Wide-Area Resource Discovery. In *14th International Symposium on High Performance Distributed Computing*, Los Alamitos, CA., July 2005. IEEE, IEEE Computer Society Press. Cited on 253

Oram A., editor. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, Sebastopol, CA., 2001. Cited on

Özsu T. and Valduriez P. *Principles of Distributed Database Systems*. Springer-Verlag, Berlin, 3rd edition, 2011. Cited on 330

Ozsu T. and Valduriez P. *Principles of Distributed Database Systems*. Prentice Hall, Upper Saddle River, N.J., 2nd edition, 1999. Cited on 62

Pai V., Aron M., Banga G., Svendsen M., Druschel P., Zwaenepoel W., and Nahum E. Locality-Aware Request Distribution in Cluster-Based Network Servers. In *8th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 205–216, New York, NY, Oct. 1998. ACM, ACM Press. Cited on 116

Panzieri F. and Shrivastava S. Rajdoot: A Remote Procedure Call Mechanism with Orphan Detection and Killing. *IEEE Transactions on Software Engineering*, 14(1): 30–37, Jan. 1988. Cited on 376

Pease M., Shostak R., and Lamport L. Reaching Agreement in the Presence of Faults.

Journal of the ACM, 27(2):228–234, Apr. 1980. Cited on 358

Perkins C., Hodson O., and Hardman V. A Survey of Packet Loss Recovery Techniques for Streaming Audio. *IEEE Network*, 12(5):40–48, Sept. 1998. Cited on 188

Peterson L., Bavier A., Fiuczynski M., Muir S., and Roscoe T. Towards a Comprehensive PlanetLab Architecture. Technical Report PDN-05-030, PlanetLab Consortium, June 2005. Cited on 121, 124

Pfleeger C. *Security in Computing*. Prentice Hall, Upper Saddle River, N.J., 3rd edition, 2003. Cited on 412, 428

Picco G., Balzarotti D., and Costa P. LighTS: A Lightweight, Customizable Tuple Space Supporting Context-Aware Applications. In *Symposium on Applied Computing*, pages 413–419, New York, NY, Mar. 2005. ACM, ACM Press. Cited on 641

Pierre G. and Steen M.van . Globule: A Collaborative Content Delivery Network. *IEEE Communications Magazine*, 44(8):127–133, Aug. 2006. Cited on 73, 83

Pierre G., Steen M.van , and Tanenbaum A. Dynamically Selecting Optimal Distribution Strategies for Web Documents. *IEEE Transactions on Computers*, 51(6):637–651, June 2002. Cited on 84

Pietzuch P. R., Shand B., and Bacon J. A Framework for Event Composition in Distributed Systems. In *Middleware 2003*, volume 2672 of *Lecture Notes in Computer Science*, pages 62–82, Berlin, June 2003. IFIP/ACM, Springer-Verlag. Cited on 651

Pike R., Presotto D., Dorward S., Flandrena B., Thompson K., Trickey H., and Winterbottom P. Plan 9 from Bell Labs. *Computing Systems*, 8(3):221–254, Summer 1995. Cited on 224, 546

Pinzari G. NX X Protocol Compression. Technical Report D-309/3-NXP-DOC, NoMachine, Rome, Italy, Sept. 2003. Cited on 106

Pitoura E. and Samaras G. Locating Objects in Mobile Computing. *IEEE Transactions on Knowledge and Data Engineering*, 13(4):571–592, July 2001. Cited on 219

Plainfosse D. and Shapiro M. A Survey of Distributed Garbage Collection Techniques. In *Proceedings International Workshop on Memory Management*, volume 986 of *Lecture Notes in Computer Science*, pages 211–249. Springer-Verlag, Berlin, Sept. 1995. Cited on 212

Plummer D. Ethernet Address Resolution Protocol. RFC 826, Nov. 1982. Cited on 209

Podling S. and Boszormenyi L. A Survey of Web Cache Replacement Strategies. *ACM Computing Surveys*, 35(4):374–398, Dec. 2003. Cited on 620

Popescu B., Steen M.van , and Tanenbaum A. A Security Architecture for Object-Based Distributed Systems. In *18th Annual Computer Security Applications Conference*. ACSA, Dec. 2002. Cited on 522

Postel J. Simple Mail Transfer Protocol. RFC 821, Aug. 1982. Cited on 177

Postel J. and Reynolds J. File Transfer Protocol. RFC 995, Oct. 1985. Cited on 144

Potzl H., Anderson M., and Steinbrink B. Linux-VServer: Resource Efficient Context Isolation. *Free Software Magazine*, (5), June 2005. Cited on 125

Pouwelse J., Garbacki P., Epema D., and Sips H. A Measurement Study of the BitTorrent Peer-to-Peer File-Sharing System. Technical Report PDS-2004-003,

Technical University Delft, Apr. 2004. Cited on 73

Pouwelse J. A., Garbacki P., Epema D. H. J., and Sips H. J. The Bittorrent P2P File-Sharing System: Measurements and Analysis. In *4th International Workshop on Peer-to-Peer Systems*, volume 3640 of *Lecture Notes in Computer Science*, pages 205–216, Berlin, Feb. 2005. Springer-Verlag. Cited on 570

Qin F., Tucek J., Sundaresan J., and Zhou Y. Rx: Treating Bugs as Allergies - A Safe Method to Survive Software Failures. In *20th Symposium on Operating System Principles*, pages 235–248, New York, NY, Oct. 2005. ACM, ACM Press. Cited on 408

Qiu L., Padmanabhan V., and Voelker G. On the Placement of Web Server Replicas. In *20th INFOCOM Conference*, pages 1587–1596, Los Alamitos, CA., Apr. 2001. IEEE, IEEE Computer Society Press. Cited on 328, 329

Rabinovich M. and Spastscheck O. *Web Caching and Replication*. Addison-Wesley, Reading, MA., 2002. Cited on 72, 617

Rabinovich M., Rabinovich I., Rajaraman R., and Aggarwal A. A Dynamic Object Replication and Migration Protocol for an Internet Hosting Service. In *19th International Conference on Distributed Computing Systems*, pages 101–113, Austin, TX, June 1999. IEEE. Cited on 331

Radia S. *Names, Contexts, and Closure Mechanisms in Distributed Computing Environments*. Ph.D., University of Waterloo, Ontario, 1989. Cited on 226

Radoslavov P., Govindan R., and Estrin D. Topology-Informed Internet Replica Placement. In *6th Web Caching Workshop*, Amsterdam, June 2001. North-Holland. Cited on 328

Raiciu C. and Rosenblum D. Enabling Confidentiality in Content-Based Publish/-Subscribe Infrastructures. Technical Report RN/05/30, Department of Computer Science, University College London, 2005. Cited on 667

Ramanathan P., Shin K., and Butler R. Fault-Tolerant Clock Synchronization in Distributed Systems. *Computer*, 23(10):33–42, Oct. 1990. Cited on 267

Ramasubramanian V. and Sirer E. G. Beehive: O(1) Lookup Performance for Power-Law Query Distributions in Peer-to-Peer Overlays. In *1st Symposium on Networked Systems Design and Implementation*, pages 99–112, Berkeley, CA, Mar. 2004a. USENIX, USENIX. Cited on 245, 570

Ramasubramanian V. and Sirer E. G. The Design and Implementation of a Next Generation Name Service for the Internet. In *SIGCOMM*, New York, NY, Aug. 2004b. ACM, ACM Press. Cited on 244

Ratnasamy S., Francis P., Handley M., Karp R., and Schenker S. A Scalable Content-Addressable Network. In *SIGCOMM*, pages 161–172, San Diego, CA, Aug. 2001. ACM. Cited on 65

Raynal M. and Singhal M. Logical Time: Capturing Causality in Distributed Systems. *Computer*, 29(2):49–56, Feb. 1996. Cited on 275

Reiter M. How to Securely Replicate Services. *ACM Transactions on Programming Languages and Systems*, 16(3):986–1009, May 1994. Cited on 446, 447

Reiter M., Birman K., and Renesse R.van . A Security Architecture for Fault-Tolerant Systems. *ACM Transactions on Computer Systems*, 12(4):340–371, Nov. 1994. Cited on 472

Rescorla E. and Schiffman A. The Secure HyperText Transfer Protocol. RFC 2660, Aug. 1999. Cited on 611

Reynolds J. and Postel J. Assigned Numbers. RFC 1700, Oct. 1994. Cited on 110

Ricart G. and Agrawala A. An Optimal Algorithm for Mutual Exclusion in Computer Networks. *Communications of the ACM*, 24(1):9–17, Jan. 1981. Cited on 285

Risson J. and Moors T. Survey of Research towards Robust Peer-to-Peer Networks: Search Methods. *Computer Networks*, 50(17):3485–3521, 2006. Cited on 66, 256, 569

Rivest R. The MD5 Message Digest Algorithm. RFC 1321, Apr. 1992. Cited on 430

Rivest R., Shamir A., and Adleman L. A Method for Obtaining Digital Signatures and Public-key Cryptosystems. *Communications of the ACM*, 21(2):120–126, Feb. 1978. Cited on 429

Rizzo L. Effective Erasure Codes for Reliable Computer Communication Protocols. *ACM Computer Communications Review*, 27(2):24–36, Apr. 1997. Cited on 399

Rodrigues L. and others . A Transparent Light-Weight Group Service. In *15th Symposium on Reliable Distributed Systems*, pages 130–139, Niagara-on-the-Lake, Canada, Oct. 1996. IEEE. Cited on 345

Rodrigues R. and Liskov B. High Availability in DHTs: Erasure Coding vs. Replication. In *4th International Workshop on Peer-to-Peer Systems*, Feb. 2005. Cited on 575

Rodriguez P., Spanner C., and Biersack E. Analysis of Web Caching Architecture: Hierarchical and Distributed Caching. *IEEE/ACM Transactions on Networking*, 21 (4):404–418, Aug. 2001. Cited on 618

Rosenblum M. and Garfinkel T. Virtual Machine Monitors: Current Technology and Future Trends. *Computer*, 38(5):39–47, May 2005. Cited on 102

Roussos G., Marsh A. J., and Maglavera S. Enabling Pervasive Computing with Smart Phones. *IEEE Pervasive Computing*, 4(2):20–26, Apr. 2005. Cited on

Rowstron A. Run-time Systems for Coordination. In Omicini A., Zambonelli F., Klusch M., and Tolksdorf R., editors, *Coordination of Internet Agents: Models, Technologies and Applications*, pages 78–96. Springer-Verlag, Berlin, 2001. Cited on 655

Rowstron A. and Druschel P. Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Middleware 2001*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350, Berlin, Nov. 2001. Springer-Verlag. Cited on 193, 218, 244

Rowstron A. and Wray S. A Run-Time System for WCL. In Bal H., Belkhouche B., and Cardelli L., editors, *Internet Programming Languages*, volume 1686 of *Lecture Notes in Computer Science*, pages 78–96. Springer-Verlag, Berlin, 1998. Cited on 658

Russello G., Chaudron M., and Steen M.van . Dynamic Adaptation of Data Distribution Policies in a Shared Data Space System. In *International Symposium on Distributed Objects and Applications (DOA)*, volume 3291 of *Lecture Notes in Computer Science*, pages 1225–1242, Berlin, Oct. 2004. Springer-Verlag. Cited on 659, 660

Russello G., Chaudron M., Steen M.van , and Bokharouss I. An Experimental Evaluation of Self-managing Availability in Shared Data Spaces. *Science of Computer Programming*, 64(2):246–262, Jan. 2007. Cited on 659, 664

Sadjadi S. and McKinley P. A Survey of Adaptive Middleware. Technical Report MSU-CSE-03-35, Michigan State University, Computer Science and Engineering, Dec. 2003. Cited on 75

Saltzer J. and Schroeder M. The Protection of Information in Computer Systems. *Proceedings of the IEEE*, 63(9):1278–1308, Sept. 1975. Cited on 453

Saltzer J., Reed D., and Clark D. End-to-End Arguments in System Design. *ACM Transactions on Computer Systems*, 2(4):277–288, Nov. 1984. Cited on 281

Sandhu R. S., Coyne E. J., Feinstein H. L., and Youman C. E. Role-Based Access Control Models. *Computer*, 29(2):38–47, Feb. 1996. Cited on 455

Saroiu S., Gummadi P. K., and Gribble S. D. Measuring and Analyzing the Characteristics of Napster and Gnutella Hosts. *ACM Multimedia Systems*, 9(2):170–184, Aug. 2003. Cited on 72

Satyanarayanan M. and Siegel E. Parallel Communication in a Large Distributed System. *IEEE Transactions on Computers*, 39(3):328–348, Mar. 1990. Cited on 546

Saxena P. and Rai J. A Survey of Permission-based Distributed Mutual Exclusion Algorithms. *Computer Standards and Interfaces*, 25(2):159–181, May 2003. Cited on 281

Schmidt D., Stal M., Rohnert H., and Buschmann F. *Pattern-Oriented Software Architecture – Patterns for Concurrent and Networked Objects*. John Wiley, New York, 2000. Cited on 75

Schneider F. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Computing Surveys*, 22(4):299–320, Dec. 1990. Cited on 277, 335, 520

Schneier B. *Secrets and Lies*. John Wiley, New York, 2000. Cited on 426

Schneier B. *Applied Cryptography*. John Wiley, New York, 2nd edition, 1996. Cited on 426, 448

Schulzrinne H. The tel URI for Telephone Numbers. RFC 3966, Jan. 2005. Cited on 615

Schulzrinne H., Casner S., Frederick R., and Jacobson V. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, July 2003. Cited on 143

Sebesta R. *Programming the World Wide Web*. Addison-Wesley, Reading, MA., 3rd edition, 2006. Cited on 591

Shapiro M., Dickman P., and Plainfosse D. SSP Chains: Robust, Distributed References Supporting Acyclic Garbage Collection. Technical Report 1799, INRIA, Rocquencourt, France, Nov. 1992. Cited on 211

Shaw M. and Clements P. A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems. In *21st International Computer Software and Applications Conference*, pages 6–13, Aug. 1997. Cited on 52

Shepler S., Callaghan B., Robinson D., Thurlow R., Beame C., Eisler M., and Noveck D. Network File System (NFS) Version 4 Protocol. RFC 3530, Apr. 2003. Cited on 229, 531

Sheth A. P. and Larson J. A. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3): 183–236, Sept. 1990. Cited on 331

Shooman M. L. *Reliability of Computer Systems and Networks: Fault Tolerance, Analysis,*

and Design. John Wiley, New York, 2002. Cited on 354

Silberschatz A., Galvin P., and Gagne G. *Operating System Concepts*. John Wiley, New York, 7th edition, 2005. Cited on 224

Singh A., Castro M., Druschel P., and Rowstron A. Defending Against Eclipse Attacks on Overlay Networks. In *11th SIGOPS European Workshop*, pages 115–120, New York, NY, Sept. 2004. ACM, ACM Press. Cited on 583

Singh A., Ngan T.-W., Druschel P., and Wallach D. S. Eclipse Attacks on Overlay Networks: Threats and Defenses. In *25th INFOCOM Conference*, pages 1–12, Los Alamitos, CA., Apr. 2006. IEEE, IEEE Computer Society Press. Cited on 583

Singhal M. and Shivaratri N. *Advanced Concepts in Operating Systems: Distributed, Database, and Multiprocessor Operating Systems*. McGraw-Hill, New York, 1994. Cited on 399

Sivasubramanian S., Pierre G., and Steen M.van . Replicating Web Applications On-Demand. In *1st International Conference on Services Computing*, pages 227–236, Los Alamitos, CA., Sept. 2004a. IEEE, IEEE Computer Society Press. Cited on 627

Sivasubramanian S., Szymaniak M., Pierre G., and Steen M.van . Replication for Web Hosting Systems. *ACM Computing Surveys*, 36(3):1–44, Sept. 2004b. Cited on 331, 620

Sivasubramanian S., Alonso G., Pierre G., and Steen M.van . GlobeDB: Autonomic Data Replication for Web Applications. In *14th International World Wide Web Conference*, pages 33–42, New York, NY, May 2005. ACM Press. Cited on 628

Sivasubramanian S., Pierre G., Steen M.van , and Alonso G. GlobeCBC: Content-blind Result Caching for Dynamic Web Applications. Technical report, Vrije Universiteit, Department of Computer Science, Jan. 2006. Cited on 629

Sivrikaya F. and Yener B. Time Synchronization in Sensor Networks: A Survey. *IEEE Network*, 18(4):45–50, July 2004. Cited on 271

Skeen D. Nonblocking Commit Protocols. In *SIGMOD International Conference on Management Of Data*, pages 133–142. ACM, 1981. Cited on 395

Skeen D. and Stonebraker M. A Formal Model of Crash Recovery in a Distributed System. *IEEE Transactions on Software Engineering*, SE-9(3):219–228, Mar. 1983. Cited on 396

Smith J. and Nair R. The Architecture of Virtual Machines. *Computer*, 38(5):32–38, May 2005. Cited on 101, 102

Snir M., Otto S., Huss-Lederman S., Walker D., and Dongarra J. *MPI: The Complete Reference – The MPI Core*. MIT Press, Cambridge, MA., 1998. Cited on 169

Speakman T., Crowcroft J., Gemmell J., Farinacci D., Lin S., Leshchiner D., Luby M., Montgomery T., Rizzo L., Tweedly A., Bhaskar N., Edmonstone R., Sumanasekera R., and Vicisano L. PGM Reliable Transport Protocol Specification. RFC 3208, Dec. 2001. Cited on 662

Specht S. M. and Lee R. B. Distributed Denial of Service: Taxonomies of Attacks, Tools, and Countermeasures. In *Int'l Workshop on Security in Parallel and Distributed Systems*, pages 543–550, Sept. 2004. Cited on 465

Spector A. Performing Remote Operations Efficiently on a Local Computer Network. *Communications of the ACM*, 25(4):246–260, Apr. 1982. Cited on 372

Srinivasan R. RPC: Remote Procedure Call Protocol Specification Version 2. RFC

1831, Aug. 1995a. Cited on 543

Srinivasan R. XDR: External Data Representation Standard. RFC 1832, Aug. 1995b. Cited on 543

Sripanidkulchai K., Maggs B., and Zhang H. Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems. In *22nd INFOCOM Conference*, Los Alamitos, CA., Mar. 2003. IEEE, IEEE Computer Society Press. Cited on 255

Stein L. *Web Security, A Step-by-Step Reference Guide*. Addison-Wesley, Reading, MA., 1998. Cited on 471

Steinder M. and Sethi A. A Survey of Fault Localization Techniques in Computer Networks. *Science of Computer Programming*, 53:165–194, May 2004. Cited on 408

Steiner J., Neuman C., and Schiller J. Kerberos: An Authentication Service for Open Network Systems. In *Winter Technical Conference*, pages 191–202. USENIX, 1988. Cited on 448

Steinmetz R. Human Perception of Jitter and Media Synchronization. *IEEE Journal on Selected Areas in Communication*, 14(1):61–72, Jan. 1996. Cited on 190

Steinmetz R. and Nahrstedt K. *Multimedia Systems*. Springer-Verlag, Berlin, 2004. Cited on 115, 184, 187

Stevens W. *UNIX Network Programming – Networking APIs: Sockets and XTI*. Prentice Hall, Englewood Cliffs, N.J., 2nd edition, 1998. Cited on 97, 167

Stevens W. *UNIX Network Programming – Interprocess Communication*. Prentice Hall, Englewood Cliffs, N.J., 2nd edition, 1999. Cited on 90, 159

Stevens W. and Rago S. *Advanced Programming in the UNIX Environment*. Addison-Wesley, Reading, MA., 2nd edition, 2005. Cited on 92

Stoica I., Morris R., Liben-Nowell D., Karger D. R., Kaashoek M. F., Dabek F., and Balakrishnan H. Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, Feb. 2003. Cited on 63, 214, 217

Stojmenovic I. Position-based Routing in Ad Hoc Networks. *IEEE Communications Magazine*, 40(7):128–134, July 2002. Cited on 291

Strauss J., Katabi D., and Kaashoek F. A Measurement Study of Available Bandwidth Estimation Tools. In *3rd Internet Measurement Conference*, pages 39–44, New York, NY, 2003. ACM Press. Cited on 621

Sugerman J., Venkitachalam G., and Lim B.-H. Virtualizing I/O Devices on VMware Workstation s Hosted Virtual Machine Monitor. In *USENIX Annual Technical Conference*, pages 1–14, Berkeley, CA, June 2001. USENIX, USENIX. Cited on 102

Sun Microsystems . *EJB 3.0 Simplified API*. Sun Microsystems, Mountain View, Calif., Aug. 2005. Cited on 485

Sun Microsystems . *Java Message Service, version 1.1*. Sun Microsystems, Mountain View, Calif., Apr. 2004a. Cited on 506, 639

Sun Microsystems . *Java Remote Method Invocation Specification, JDK 1.5*. Sun Microsystems, Mountain View, Calif., 2004b. Cited on 145

Jini. *Jini Technology Starter Kit, Version 2.1*. Sun Microsystems, Palo Alto, CA, Oct. 2005. Cited on 527, 639

Sundararaman B., Buy U., and Kshemkalyani A. D. Clock Synchronization for Wireless Sensor Networks: A Survey. *Ad-Hoc Networks*, 3(3):281–323, May 2005.

Cited on 271

Szymaniak M., Pierre G., and Steen M.van . Scalable Cooperative Latency Estimation. In *10th International Conference on Parallel and Distributed Systems*, pages 367–376, Los Alamitos, CA., July 2004. IEEE, IEEE Computer Society Press. Cited on 293

Szymaniak M., Pierre G., and Steen M.van . A Single-Homed Ad hoc Distributed Server. Technical Report IR-CS-013, Vrije Universiteit, Department of Computer Science, Mar. 2005. Cited on 118

Szymaniak M., Pierre G., and Steen M.van . Latency-driven replica placement. *IPSJ Digital Courier*, 2:561–572, 2006. Cited on 328

Taiani F., Fabre J.-C., and Killijian M.-O. A Multi-Level Meta-Object Protocol for Fault-Tolerance in Complex Architectures. In *International Conference on Dependable Systems and Networks*, pages 270–279, Los Alamitos, CA., June 2005. IEEE Computer Society Press. Cited on 514

Tam D., Azimi R., and Jacobsen H.-A. Building Content-Based Publish/Subscribe Systems with Distributed Hash Tables. In *1st Int'l Workshop on Databases, Information Systems and Peer-to-Peer Computing*, volume 2944 of *Lecture Notes in Computer Science*, pages 138–152, Berlin, Sept. 2003. Springer-Verlag. Cited on 643

Tan S.-W., Waters G., and Crawford J. A Survey and Performance Evaluation of Scalable Tree-based Application Layer Multicast Protocols. Technical Report 9-03, University of Kent, UK, July 2003. Cited on 196

Tanenbaum A. *Computer Networks*. Prentice Hall, Upper Saddle River, N.J., 4th edition, 2003. Cited on 139, 369

Tanenbaum A. and Woodhull A. *Operating Systems, Design and Implementation*. Prentice Hall, Englewood Cliffs, N.J., 3rd edition, 2006. Cited on 224, 535

Tanenbaum A., Mullender S., and Renesse R.van . Using Sparse Capabilities in a Distributed Operating System. In *6th International Conference on Distributed Computing Systems*, pages 558–563, Cambridge, MA, May 1986. IEEE. Cited on 474

Tanenbaum A., Renesse R.van , Staveren H.van , Sharp G., Mullender S., Jansen J., and Rossum G.van . Experiences with the Amoeba Distributed Operating System. *Communications of the ACM*, 33(12):46–63, Dec. 1990. Cited on 452

Tanisch P. Atomic Commit in Concurrent Computing. *IEEE Concurrency*, 8(4):34–41, Oct. 2000. Cited on 389

Tartalja I. and Milutinovic V. Classifying Software-Based Cache Coherence Solutions. *IEEE Software*, 14(3):90–101, May 1997. Cited on 347

Tel G. *Introduction to Distributed Algorithms*. Cambridge University Press, Cambridge, UK, 2nd edition, 2000. Cited on 259, 294

Terry D., Demers A., Petersen K., Spreitzer M., Theimer M., and Welsh B. Session Guarantees for Weakly Consistent Replicated Data. In *3rd International Conference on Parallel and Distributed Information Systems*, pages 140–149, Los Alamitos, CA., Sept. 1994. IEEE, IEEE Computer Society Press. Cited on 322, 325, 327

Terry D., Petersen K., Spreitzer M., and Theimer M. The Case for Non-transparent Replication: Examples from Bayou. *IEEE Data Engineering*, 21(4):12–20, Dec. 1998. Cited on 322

Thomas R. A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases. *ACM Transactions on Database Systems*, 4(2):180–209, June 1979.

Cited on 345

TIBCO . *TIB/Rendezvous Concepts, Release 7.4*. TIBCO Software Inc., Palo Alto, CA, July 2005. Cited on 74, 642

Tolia N., Harkes J., Kozuch M., and Satyanarayan M. Integrating Portable and Distributed Storage. In *3rd USENIX Conference on File and Storage Technologies*, Berkeley, CA, Mar. 2004. USENIX, USENIX. Cited on 566

Tolksdorf R. and Rowstron A. Evaluating Fault Tolerance Methods for Large-scale Linda-like systems. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, volume 2, pages 793–800, June 2000. Cited on 664

Towsley D., Kurose J., and Pingali S. A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols. *IEEE Journal on Selected Areas in Communication*, 15(3):398–407, Apr. 1997. Cited on 378

Tripathi A., Karnik N., Vora M., Ahmed T., and Singh R. Mobile Agent Programming in Ajanta. In *19th International Conference on Distributed Computing Systems*, pages 190–197, Austin, TX, May 1999. IEEE. Cited on 459

Turek J. and Shasha S. The Many Faces of Consensus in Distributed Systems. *Computer*, 25(6):8–17, June 1992. Cited on 364, 367

Umar A. *Object-Oriented Client/Server Internet Environments*. Prentice Hall, Upper Saddle River, N.J., 1997. Cited on 60

UPnP Forum . UPnP Device Architecture Version 1.0.1, Dec. 2003. Cited on

Renesse R.van , Birman K., and Vogels W. Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining. *ACM Transactions on Computer Systems*, 21(2):164–206, May 2003. Cited on 81

Steen M.van , Hauck F., Homburg P., and Tanenbaum A. Locating Objects in Wide-Area Systems. *IEEE Communications Magazine*, 36(1):104–109, Jan. 1998. Cited on 218

Vasudevan S., Kurose J. F., and Towsley D. F. Design and Analysis of a Leader Election Algorithm for Mobile Ad Hoc Networks. In *12th International Conference on Network Protocols*, pages 350–360, Los Alamitos, CA., Oct. 2004. IEEE, IEEE Computer Society Press. Cited on 297, 299

Veiga L. and Ferreira P. Asynchronous Complete Distributed Garbage Collection. In *19th International Parallel & Distributed Processing Symposium*, Los Alamitos, CA., Apr. 2005. IEEE, IEEE Computer Society Press. Cited on 212

Velazquez M. A Survey of Distributed Mutual Exclusion Algorithms. Technical Report CS-93-116, University of Colorado at Boulder, Sept. 1993. Cited on 281

Vetter R., Spell C., and Ward C. Mosaic and the World-Wide Web. *Computer*, 27(10): 49–57, Oct. 1994. Cited on 589

Vitek J., Bryce C., and Oriol M. Coordinating Processes with Secure Spaces. *Science of Computer Programming*, 46(1-2), 2003. Cited on 669

Vogels W. Tracking Service Availability in Long Running Business Activities. In *1st International Conference on Service Oriented Computing*, volume 2910 of *Lecture Notes in Computer Science*, pages 395–408, Berlin, Dec. 2003. Springer-Verlag. Cited on 369

Voulgaris S. and Steen M.van . Epidemic-style Management of Semantic Overlays for Content-Based Searching. In *11th International Conference on Parallel and Distributed*

Computing (Euro-Par), volume 3648 of *Lecture Notes in Computer Science*, pages 1143–1152, Berlin, Sept. 2005. Springer-Verlag. Cited on 255

Voulgaris S., Rivière E., Kermarrec A.-M., and Steen M.van . Sub-2-Sub: Self-Organizing Content-Based Publish and Subscribe for Dynamic and Large Scale Collaborative Networks. In *5th International Workshop on Peer-to-Peer Systems*, Feb. 2006. Cited on 643, 646

Voydock V. and Kent S. Security Mechanisms in High-Level Network Protocols. *ACM Computing Surveys*, 15(2):135–171, June 1983. Cited on 432

Wah B. W., Su X., and Lin D. A Survey of Error-Concealment Schemes for Real-Time Audio and Video Transmissions over the Internet. In *Int'l Symp. Multimedia Softw. Eng.*, pages 17–24, Los Alamitos, CA., Dec. 2000. IEEE, IEEE Computer Society Press. Cited on 188

Wahbe R., Lucco S., Anderson T., and Graham S. Efficient Software-based Fault Isolation. In *14th Symposium on Operating System Principles*, pages 203–216, Asheville, North Carolina, Dec. 1993. ACM. Cited on 460

Waldo J. Remote Procedure Calls and Java Remote Method Invocation. *IEEE Concurrency*, 6(3):5–7, July 1998. Cited on 502

Walfish M., Balakrishnan H., , and Shenker S. Untangling the Web from DNS. In *1st Symposium on Networked Systems Design and Implementation*, pages 225–238, Berkeley, CA, Mar. 2004. USENIX, USENIX. Cited on 243

Wallach D. A Survey of Peer-to-Peer Security Issues. In *Int'l Symp. Softw. Security*, volume 2609 of *Lecture Notes in Computer Science*, pages 42–57, Berlin, Nov. 2002. Springer-Verlag. Cited on 583

Wallach D., Balfanz D., Dean D., and Felten E. Extensible Security Architectures for Java. In *16th Symposium on Operating System Principles*, pages 116–128, St. Malo, France, Oct. 1997. ACM. Cited on 462, 465

Wang C., Carzaniga A., Evans D., and Wolf A. L. Security Issues and Requirements for Internet-Scale Publish-Subscribe Systems. In *35th Hawaii International Conference on System Sciences*, volume 9, pages 303–310. IEEE, Jan. 2002. Cited on 666

Wang H., Lo M. K., and Wang C. Consumer Privacy Concerns about Internet Marketing. *Communications of the ACM*, 41(3):63–70, Mar. 1998. Cited on

Watts D. *Small Worlds, The Dynamics of Networks between Order and Randomness*. Princeton University Press, Princeton, NJ, 1999. Cited on 255

Wessels D. *Squid: The Definitive Guide*. O'Reilly & Associates, Sebastopol, CA., 2004. Cited on 601, 619

Wieringa R. and Jonge W.de . Object Identifiers, Keys, and Surrogates–Object Identifiers Revisited. *Theory and Practice of Object Systems*, 1(2):101–114, 1995. Cited on 207

Wiesmann M., Pedone F., Schiper A., Kemme B., and Alonso G. Understanding Replication in Databases and Distributed Systems. In *20th International Conference on Distributed Computing Systems*, pages 264–274, Taipei, Taiwan, Apr. 2000. IEEE. Cited on 306

Wollrath A., Riggs R., and Waldo J. A Distributed Object Model for the Java System. *Computing Systems*, 9(4):265–290, Fall 1996. Cited on 499, 512

Wolman A., Voelker G., Sharma N., Cardwell N., Karlin A., and Levy H. On the

Scale and Performance of Cooperative Web Proxy Caching. In *17th Symposium on Operating System Principles*, pages 16–31, Kiawah Island, SC, Dec. 1999. ACM. Cited on 618

Wu D., Hou Y., Zhu W., Zhang Y., and Peha J. Streaming Video over the Internet: Approaches and Directions. *IEEE Trans. Circuits & Syst. Video Techn.*, 11(1):1–20, Feb. 2001. Cited on 186

Yang B. and Garcia-Molina H. Designing a Super-Peer Network. In *19th International Conference on Data Engineering*, pages 49–60, Los Alamitos, CA., Mar. 2003. IEEE, IEEE Computer Society Press. Cited on 70

Yang M., Zhang Z., Li X., and Dai Y. An Empirical Study of Free-Riding Behavior in the Maze P2P File-Sharing System. In *4th International Workshop on Peer-to-Peer Systems*, Lecture Notes in Computer Science, Berlin, Feb. 2005. Springer-Verlag. Cited on 72

Yellin D. Competitive Algorithms for the Dynamic Selection of Component Implementations. *IBM Systems Journal*, 42(1):85–97, Jan. 2003. Cited on 78

Yu H. and Vahdat A. Efficient Numerical Error Bounding for Replicated Network Services. In Abbadi A. E., Brodie M. L., Chakravarthy S., Dayal U., Kamel N., Schlageter G., and Whang K.-Y., editors, *26th International Conference on Very Large Data Bases*, pages 123–133, San Mateo, CA., Sept. 2000. Morgan Kaufman. Cited on 339

Yu H. and Vahdat A. Design and Evaluation of a Conit-Based Continuous Consistency Model for Replicated Services. *ACM Transactions on Computer Systems*, 20(3): 239–282, 2002. Cited on 308, 309, 622

Zhang C. and Jacobsen H.-A. Resolving Feature Convolution in Middleware Systems. In *19th Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 188–205, New York, NY, Oct. 2004. ACM, ACM Press. Cited on 78

Zhao B., Huang L., Stribling J., Rhea S., Joseph A., and Kubiatowicz J. Tapestry: A Resilient Global-Scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communication*, 22(1):41–53, Jan. 2004. Cited on 244

Zhao F. and Guibas L. *Wireless Sensor Networks*. Morgan Kaufman, San Mateo, CA., 2004. Cited on

Zhuang S. Q., Geels D., Stoica I., and Katz R. H. On Failure Detection Algorithms in Overlay Networks. In *24th INFOCOM Conference*, Los Alamitos, CA., Mar. 2005. IEEE, IEEE Computer Society Press. Cited on 368

Zogg J.-M. GPS Basics. Technical Report GPS-X-02007, UBlox, Mar. 2002. Cited on 265, 267

Zwicky E., Cooper S., Chapman D., and Russell D. *Building Internet Firewalls*. O'Reilly & Associates, Sebastopol, CA., 2nd edition, 2000. Cited on 456