The Snowflake Distributed System

Thesis Proposal

Jon Howell

Department of Computer Science
Dartmouth College
Hanover, NH 03755-3510
jonh@cs.dartmouth.edu

December 8, 1998

Abstract

We describe an emerging problem in globally distributed operating systems: the conflict between user requirements and administrative requirements. Then we describe the key to the solution: user-level resource aggregation. Next we outline the requirements of such a solution, and then we describe our specific implementation of the solution, which we call Snowflake. Finally, we discuss the metrics we will use to evaluate the Snowflake solution, and how to construct an experiment and measure the system to accomplish that evaluation.

1 The problem: a conflict between ease of use and ease of administration

In times past, an operating system was defined as "hardware abstraction and resource management" [SS94a, Tan87]. That is, in some sense, a **single-computer** operating system was at the junction of users' and administrators' experience. Protection provided a mechanism for administration, and abstraction provided ease of use. Administrators were arbiters between users; the conflict was between users.

As operating systems grew into **distributed systems** (local-area clusters), the challenge was finding suitable abstractions that harnessed larger systems into an easy-to-use environment. The conflict was between users and the unruly resources they were trying to find and use.

As we enter a world in which more and more resources participate in a **global network**, a conflict is forming between users and administrators. The many users to one administrator relationship is generalizing into a many-to-many relationship. Because a single administrator no longer has total responsibility for a user's experience, the emerging conflict is between ease of use (of separately administered resources) and ease of administration (of widely used resources).

1.1 What is ease of use?

A user's experience is greatly eased when resources are presented to him on his own terms. It is not enough that he can name any resource in the world; he should be able to give that resource a name that he finds mnemonic. Personal computers have exemplified this trend toward customizability; the general concept is that a user should see any resource in the world as part of his personal

single-system image. This section describes the problems inherent in making a global single-system image personal and easy-to-use.

Naming

1. **Per-user naming.** A user wants to organize his resources in a consistent way that makes sense to him. He should be able to do this regardless of where he found each resource.

2. **Location transparency.** Resource names should not expose their source. In general, the user does not care where he found something, he cares what it can do for him.

Sharing and access control

1. **Single sign-on.** The user has access to some broad set of resources around the world. He should be able to access the entire set of them in a consistent way. He should prove his identity once, and then have access to all of his resources anywhere they may live.

2. **Untrusted code.** The principle of least privilege suggests that he should not run every program with all of his rights intact, lest some program violate his trust and misuse his resources. He needs a way to conveniently limit rights when running untrusted code.

3. **Sharing.** He will want to share his resources with other people. He should be able to conveniently define groups of principles and the set of rights he will confer on them. An important requirement is that he understand what he is doing; a great security mechanism is useless if he cannot understand it and simply defeats it just to get a job done.

1.2 What is ease of administration?

An administrator has a responsibility to configure resources, make them available to users, and protect them from adversaries. To accomplish these objectives, she needs a certain amount of control over how she does her job. Systems shared among multiple users (that is, non-PCs) tend to err on the side of providing the administrator with more control than she needs. The extra control limits users' freedom, interfering with ease of use as described above. And it means that users pester the system administrator to perform tasks that should not require her attention. To what administrative concerns must our design be sensitive?

Resource configuration

Most users use most resources in some default configuration. The system administrator can preempt help-desk questions by supplying reasonable default configurations. However, some users invariably want to reconfigure some resources; they should be able to do this as well without hassling the administrator. Finally, it would be nice if the system administrator could avoid cleaning up after users who deviate from the defaults, then complain that "it doesn't work."

Autonomy

The administrator's boss gives her certain requirements that she must implement as system policy. But when she makes resources available to other users, those users will no doubt want to mix and match resources with those from other sites. So she needs autonomy from other sites to implement the policies required by her boss.

1. She does not care if users use resources from other sites, but when they use her resources, she gets to define the local policies. Stated another way, a user's single-system image should not be molded into the shape of her local hierarchy, but the policies handed down by her superiors still must apply to those resources she hands out.

2. When users use resources from other sites, it would be nice for the administrator to tell and/or control how much the inter-site interaction affects her. That is, she wants to serve her users, but not to provide support for resources from other sites.

Security

To protect resources from adversaries, the administrator must establish standards for authentication, and must be able to control leakage of privileges.

1. When using resources at her site, users must to conform to her standards of satisfactory authentication.

2. She will want to limit users' ability to share rights to resources from her site, according to the local policy governing those resources.

1.3 The conflict

I claimed that there is a conflict between ease of use and ease of administration. It is not an inherent conflict. Rather, traditional systems, which consider abstraction part of the job of the operating system (specifically, under the administrator's control), manifest this conflict.

The problem is that ease of use (single-system images) conflict with existing administrative mechanisms for implementing local policies. For example, users cannot use NFS or AFS to remotely access their files in another administrative domain; that mechanism requires a system administrator's privilege. A UNIX user cannot create a group on the fly to reflect the set of people he is working with this week; groups are a system administrator-controlled mechanism. Often programs and their configuration files are installed in read-only locations; depending on the authorship of the program, it is either inconvenient or impossible for an ordinary user to reconfigure the program. In our world, it is difficult for single-system images to span administrative domains. Because of needlessly inconvenient mechanisms, users cannot perform many tasks that they would need to build a personal single-system image.

The solution is to somehow make "single-system images" orthogonal to "multiple administrative domains." I posit that by providing the right model and mechanisms, users will have the tools and freedom they need to build single-system images, while administrators will still be able to maintain their administrative domains.

Where does the conflict come from? In each of the examples I mentioned above, there was some level of abstraction or resource aggregation that, because of how the relevant administrative mechanisms were built, can only be controlled by the administrator. But fundamentally, the administrator does not require that control.

In this proposal, we will describe our solution to the problem outlined above. In Section 2, we describe the philosophical idea behind our solution. Section 3 details specific components needed to implement that idea. Section 4 describes how our current experiment implements those details. In Section 5 we discuss the terms on which we will evaluate our solution, and in Section 6 we describe specific measurements we can make to support our evaluation. Section 7 proposes a timeline for the completion of this work. Finally, Section 8 discusses related work and how it has and will apply to our project.

2 A solution: user-level resource aggregation

The solution, then, is to rearrange the system so that abstraction, and specifically resource aggregation, are generally performed under user control. This solution is sketched in Figure 1. Where traditional systems leave no room for user decisions about resource distribution and aggregation, we propose to insert a layer between application software and administrator-controlled operating system services in which users control the resource distribution and aggregation. In this structure, all applications use a uniform distribution mechanism, and that mechanism is under the ultimate control of the end user.

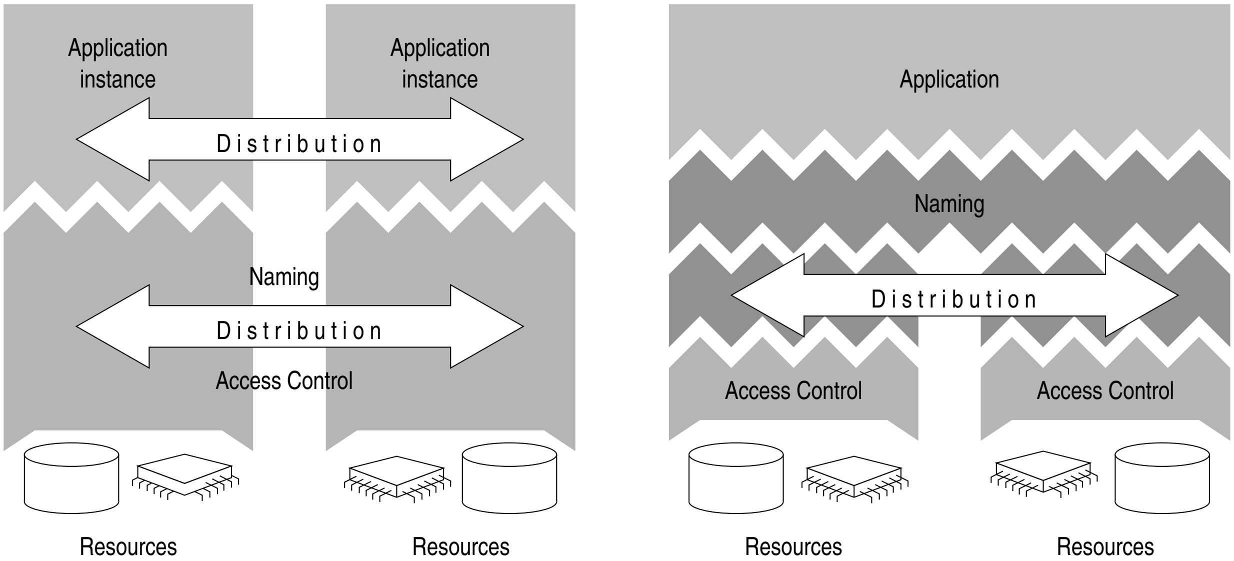

Figure 1: (a) Traditional systems either provide distribution in the application, under programmer control, or in the operating system, under system administrator control. (b) User-level resource aggregation means that distribution and aggregation of resources is performed under user control.

Resource aggregation is the process of making lots of resources work together to present a unified, easily comprehended and used interface. It is fundamentally a problem of creating a higher-level abstraction that hides uninteresting details. This abstraction must consider issues of sharing and security, because users aggregate resources that are supplied by some other entity, a system administrator or another user.

User-level means guaranteeing that such abstractions are under the control of the user. That is, because the abstractions are merely a tool to make the resources easier to use and manipulate, there is no reason the end-user of the resources should not have ultimate control over the abstraction tool. Certainly administrators will provide default abstractions for those users who do not have special needs, and administrators will impose a few mandatory abstractions because they are necessary to define and enforce a desired protection policy. But the crux of the "user-level resource aggregation" concept is that at all points, the design of the system attempts to reserve for the user control over how resources are aggregated.

3 Details of the solution

What does it mean for a design to reserve aggregation control for the user? It means that the system structure makes it easy to create applications and administrative tools that leave aggregation to users, and such applications and tools make it easy to administer a system in a way that leaves aggregation to users. Specifically, we define a model that provides a layer of user-controlled indirection for naming, containment, and sharing resources.

3.1 Naming & Containers

First, we consider the issue of naming distributed resources and specifying containment relationships.

3.1.1 Naming

As many resources as possible in the system are named via a uniform naming scheme, administered at user-level. Programs access all resources using names under the user's control. Therefore, by altering those bindings, the user has control over the behavior of any program.

Naming is decoupled from storage, meaning that the name of an object gives no hint as to where the object is stored. Type is decoupled from storage and naming as well: objects of different types may share the same storage medium and occupy the same corner of the namespace. This contrasts with Unix, where mount points ensure that subtrees of objects sharing a common name prefix also share the same type or storage class.

A Challenge: Namespace Traversal

Decoupling names from storage creates difficulties in traversing the namespace. In traditional systems, one can change the storage of a data object by renaming the object; the destination name defines the new storage, and the move command can recursively descend all of the data objects associated with the given name.

However, we may need a convention by which one name refers to how to use a given resource, but another name is used to identify the storage resources that back the given resource. The latter name would be used for traversal and data movement operations.

Explicit Interfaces

Instead of declaring that "everything is a file," but providing no mechanism for what type of file something is, resources publish their adherence to explicit interfaces, of which just one is the `read, write` file interface.

Explicit interfaces allow tools to discover whether they will work with a given resource. They allow a resource to implement multiple interfaces, so that it can be accessed by a wide variety of tools.

3.1.2 Containers

In the system we have described so far, a user is free to name resources from around the world according to his preference. A name does not have to reflect where a resource is stored. In contrast to conventional filesystems, there is no requirement that a namespace be a directed acyclic graph or tree. There is no sense of "up" and "down" in the namespace. Therefore, specifying a new object's name does not provide enough information to specify its storage location. We need an explicit concept of container to replace the "natural" one that occurs in filesystems.

A container defines a collection of resources that can be appended to, copied as a unit, or traversed. When we create one file in the same container as another, we expect it to be stored "near" the first, on a storage medium with the same properties. We make this concept concrete by defining explicit containers.

3.1.3 A Challenge: Persistence

A resource can have an interface richer than the typical file interface (read and write bytes). Thus we need some containers that can store resources of arbitrary type. Containers should also offer varying degrees of persistence for the resources stored within.

3.1.4 A Challenge: Type (class) evolution

Depending on how containers are implemented, one potential challenge will be to allow object types to evolve, so that object data can be preserved while object implementations are upgraded. This is particularly relevant to persistent objects, where object lifetimes can span many software development cycles.

3.2 Resource Sharing

Resource sharing is governed by one design element and one general philosophy:

1. As an explicit design element, the system is not aware of who is an "administrator" and who is a "client." This difference is made manifest only in which peers are exporting resources to which others.

2. As a philosophy, users acting in the role of "administrator" are responsible to export resources in the barest form possible. That is, abstractions should be supplied only when necessary to enforce a desired policy. For example, an administrator might export raw RAM pages instead of instances of a process abstraction. This gives the recipient user the most freedom in how she might aggregate the resources (RAM) into her single-system image: as virtual memory, perhaps, or used as a disk cache. This philosophy cannot be enforced by the system, but should be employed by users to make the system most effective.

By analogy, I would rather you shared with me eggs and sugar, rather than only allowing me to borrow cookies. The raw ingredients give me more freedom to create my personal view of the world: angelfood cake.

3.2.1 Security and Authentication Model

The administrator-user relationship is reduced to a generalized peer-to-peer relationship where one peer happens to have something the other wants. Therefore, it is clear that we require an interface by which peers can be certain of the identity of a peer with whom they are communicating. In the common case, an object exporting resources on behalf of an administrator verifies the authenticity of a client's identity, then decides (using some local authorization policy) whether the request may be granted.

4 Our implementation of the solution

The previous section describes the requirements of a solution to the problem I introduced. This section describes a specific implementation of such a solution, an experiment called *Snowflake*. Because our experiment needs to run in many different administrative domains, I chose to implement our prototype as a middleware layer above native operating systems. The implementation environment is Java.

Java is not the ideal implementation environment. It provides convenient services, such as Remote Method Invocation and object serialization. Because it is implemented on top of a conventional operating system, it is easier to debug, and it was fairly easy to implement a persistent object store using operating system functionality. However, a production implementation should happen at a lower level, for efficiency. Because Snowflake provides the services of a conventional operating system in a nonconventional way, the presence of the conventional operating system is redundant. In fact, we have discovered that Java's conventional implementation gives it traits that are detrimental to a novel design like Snowflake.

4.1 Naming & Containers

The naming interface is a simple Java interface. Implementations are supported with a raft of classes that implement various name mapping and lookup techniques. Names must ultimately resolve to some low-level "address."[1] In Snowflake, Java Remote Method Invocation URLs currently serve as low-level addresses.

The Container interface is also reified as a Java interface. Containers are implemented by instantiating objects in a Java Virtual Machine. We have developed a technique for making a JVM, and therefore the objects residing in it, persistent [How98]. This provides our system with persistent storage for long-lived objects of arbitrary type.

4.2 Sharing & Protection

Snowflake will define a concept of principal (an identity on behalf of which objects make requests), and authentication interfaces by which objects verify each other's identity.

Principals. In traditional systems, administrators define the complete set of principals (e.g., users and groups). This limits the scalability of traditional security mechanisms across administrative domains. In Snowflake, any user should be able to define principles that represent a group of other principles, for use when the user is exporting resources, or than represent a restricted set of his own privileges, for use when running untrusted programs.

Authentication Interfaces. Several schemes for satisfying demands for authentication will be available, and Snowflake will define an interface that captures the commonality between them. There will be cryptographic methods for initially verifying principal identities across untrusted links, such as network connections. There will be faster methods as an optimization, such as trusted connections (either same-machine or over cryptographically secure network connections) that guarantee that the principals on either end of the connection have not changed since it was opened.

Distributed "key" management is the default mode of the system. This allows arbitrary users to define amongst themselves their hierarchy of "administrators" and "clients." Our interfaces will

[1] An address is just another kind of name, but the term "address" is used to indicate that these low-level names do not make sense in the context of those in the level above.

define a web of trust, a la `PGP` or `ssh`. An administrator may enforce central key management (analogous to `Kerberos`) by requiring that client authentication be verified (signed) by a key distribution center; in effect, the web of trust becomes the short, fat tree of trust.

This description of the sharing and security model is intentionally vague. The related work section discusses several security models that may be applicable to this work, but I have not yet had an opportunity to experiment with security issues. A system like Snowflake must consider security issues to prove palatable to system administrators.

5 Evaluation of the solution

My problem is inherently a qualitative one, in that I am proposing new functionality: users will be able to enjoy single-system images that span administrative domains. As such, one measure of the merit of this work is how well I have achieved that vision. It will also be important to quantitatively evaluate the performance of our solution, to try to discover if the cost of our solution is worth the qualitative improvement.

5.1 Qualitative evaluation

The success of my solution must be measured in both positive and negative space: did I solve the problem, and did I avoid introducing new ones?

5.1.1 Did I solve the problem?

My evaluation will answer the following questions: Can users build single-system images that give them convenient, transparent access to resources from multiple administrative domains? Can administrators effectively manage local resources autonomously in this system? Is it easy for programmers to write applications and administrative tools that fit into the system and preserve the previous two properties?

Another important measure is the applicability of our solution. If I have merely eliminated the symptoms for one application, then I have not solved anything: there are already many examples of applications with custom protocols that can span administrative domains. Here is an analogy: In Unix, programs that use `stdin` and `stdout` almost automatically are composable tools, making the Unix toolbox philosophy apply easily to a wide variety of applications. Do the Snowflake API and conventions help programs easily become scalable across administrative domains?

5.1.2 Did I introduce new problems?

My evaluation should also investigate what functionality was lost or altered. This only makes sense in contrast to an existing standard. The risk with this evaluation is that I may not clearly state which parts of the standard (e.g. "the POSIX source code interface") are important to maintain, which parts are artifacts we can ignore, and which parts are in fact symptoms of the problem we are trying to solve. Or, the reader may disagree with my assessment of the value of parts of a standard. But a standard can be useful as a starting point or a justification for enumerating our universe of assumptions. In my evaluation, I will try to discover what features have been lost relative to the conventional Unix environment.

5.2 Quantitative evaluation

Traditionally, performance is evaluated by comparing runtimes between implementations of a common interface. For application benchmarks, the interface is the application itself, and the performance is the time from input to output of the program. For microbenchmarks, we measure the performance of some lower-level interface.

Microbenchmarks do not help us compare systems with differing low-level interfaces, because there is no common interface whose implementation we can compare between systems. Likewise, because the nature of our solution is to change interfaces, the only point of performance measurement is far above the location of the system change. Therefore, performance measurements will be crude at best: large pieces of our experimental apparatus (such as the choice of the Java environment) are likely to substantially affect system performance. This makes difficult comparisons that highlight the interface changes.

6 Measurement

The previous section described the standards by which I will evaluate my work. This section describes how I will measure the work to produce those evaluations.

This project is an experiment. The control group consists of conventional systems such as Unix. The experimental group consists of distributed systems built from software based on a modified API, runtime middleware to support that API, and philosophies and conventions that guide the construction of applications. See Figure 2. As we have mentioned, we are evaluating a change that appears to the users of the system. But to have a complete system to measure, we need to produce some applications, to see how they provide the desired experiences to users.

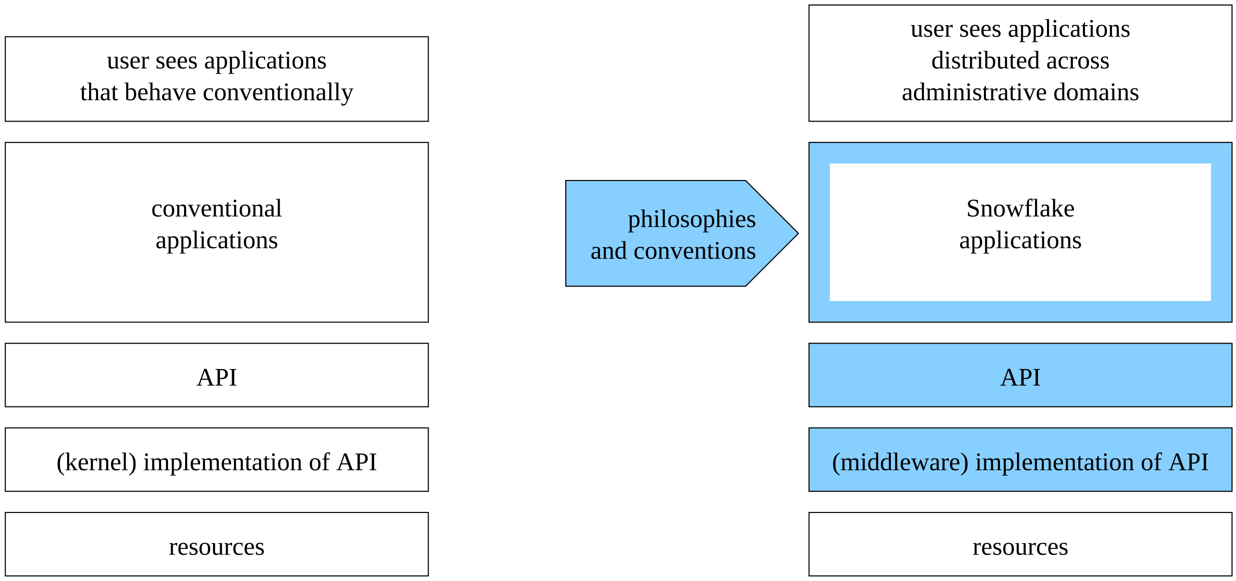

Figure 2: (a) The control group consists of conventional systems. (b) The experimental group is injected with philosophies and conventions, a new API, and an implementation of that API.

So, I will also write some applications to the Snowflake API and specifications. The applications themselves are not part of the solution, but instead part of the "experimental apparatus." What sorts of applications will highlight the interesting outputs? We are most interested in applications that use resources from across multiple administrative domains, and in applications that implement such resources. Some examples:

Development environment. The Java compiler is moved "into" the Snowflake namespace. Software development is not typically an application distributed across administrative domains, but we are doing this to make the environment self-hosting, to ensure that it is used and abused. (However, we may be able to build distributed development tools that are useful for truly distributed software development efforts, such as free-software development.)

E-mail and messaging client. I propose to build a communications client that understands e-mail and other electronic messaging protocols. Naturally, if one uses multiple administrative domains, a communications tool is the first application that would naturally benefit from distribution across those domains. Most existing tools cannot handle multiple administrative domains at all. Some have ad-hoc features that support multiple administrative domains, such as Pine's mail folder collections. Our tool will rely on the general Snowflake distribution support to provide consistent, seamless access to communications and storage resources in multiple domains.

Network management tool. Another potential application would query and manipulate resources using protocols such as SNMP. It would be designed to help an administrator observe and manage resources in his domain. A variant of this tool might help users find useful resources in a given domain.

Personal database. Personal organizer applications tend to collect snippets of data in simple databases. Users often find themselves organizing information from different aspects of their lives, as they participate in tasks at different administrative domains. Within any given department, a user might wish to share organizer data (such as appointments or contact information) with other users.

7 Timeline

The following table summarizes my goals for the Snowflake project.

Completed	Java persistence mechanism, namespace interface
Fall 1998	javac reads and writes "files" in persistent object repository. Prepare paper for Summer Usenix 1999.
Winter 1999	Create tools to manage persistent object repository; fill out development environment. Refine namespace and container interfaces with new experience.
Spring 1999	Work on e-mail and messaging client, built using tools in development environment. Begin work on security issues. Write paper to submit to SOSP. [Take AI class]
Summer 1999	[Term off to work at Sun?]
Fall 1999	Create database tool, network management tool, or other applications. Continue addressing security issues. Begin writing evaluations that compare the programming, user, and administrative environment of Snowflake with conventional systems.
Winter/Spring 2000	Continue evaluation and writing results.

8 Related Work

My proposal draws on concepts from many areas of distributed systems research. This section identifies several categories of related work. We begin in Sections 8.1 and 8.2, examining microkernels and kernel-extension technologies and the problems they try to solve, and we consider how Snowflake approaches those problems. Next, in Section 8.3, we look at user-level abstractions implemented in traditional systems. After that, we examine some single-system image clusters in Section 8.4 and some approaches to the worldwide single-system-image problem in Section 8.5. Then we explore parallel systems and distributed languages in Sections 8.6 and 8.7, which are concerned with scaling up to handle large parallel jobs, but that also provide insight into distributed systems in general. Section 8.8 is an overview of systems that provide middleware infrastructure for building distributed systems. I follow that with a discussion of distributed filesystems in Section 8.9, which attack the data-distribution component of the distributed system problem. Then, in Section 8.10, we look at the problems of mobility and network disconnection, which become relevant in wide-area distributed systems. Process migration, discussed in Section 8.11, is a common technique used in distributed systems implementations. In Section 8.12, we explore a variety of approaches to and uses for persistence. Section 8.13 gives an overview of security problems, with a focus on those related to wide area systems. Finally, Section 8.14 discusses how object-orientation has been applied to structure distributed and extensible systems.

8.1 Microkernels

Microkernels minimize the functionality implemented in the kernel, moving most functionality into user-mode processes. The processes may be administrator-owned and possibly large grained. The basic model, however, lends itself to a structure with user-level control over abstractions. Those presented here are roughly organized from lightest to heaviest weight kernel.

The SPACE project proposed a 'kernel-less' system in which events handled by traditional kernel functions, such as interrupts and device control, are reflected out of the kernel by translating those operations into IPC primitives [PB95].

The PANDA project was a Single-Address-Space Operating System (SASOS) that put most functionality out of kernel mode to achieve flexibility as well as performance. It is aimed at running parallel applications on NUMA clusters [ABB+93a, ABB+94].

For a time, the Exokernel group held the philosophy that the OS kernel boundary could also be the boundary between protection (implemented in kernel mode) and abstraction (implemented in user-mode shared libraries) [EK95, MK97, KEG+97]. They have since backed away from that position, I believe, because they discovered that abstraction and protection can be intertwined. Sometimes you need to specify protection in terms of an abstraction, like protecting files in a filesystem, rather than blocks on a disk.

Thekkath and Levy proposed a system for efficiently reflecting exceptions to user-level [TL94]. Although this project was not meant as a standalone microkernel, its strategy of reflecting hardware events into user-level as OS IPC events is similar to that used by the preceding three projects.

Liedtke's L4 microkernel champions its high-speed interprocess communication, which enables systems to be factored into smaller protection domains without suffering the unreasonable communication overhead that has long been associated with microkernels [Lie95, LES+97, HHL+97, Ber92].

Grasshopper is a general-purpose microkernel with a flexible memory model. It allows mappings ranging from traditional virtual memory to SASOS. Its memory mapping facility is used to provide pervasive services such as persistence [DdBF+94, LRD95].

The Flex project advocates a model of Recursive Virtual Machines [FS96, FHL+96]. This is facilitated by a virtual machine interface in which each resource (memory, CPU scheduling) can be administered hierarchically. While a user may only be given some fraction of the resources owned by an administrator, the user still retains the same level of expressive power to redistribute those resources.

Although it evolved from a monolithic architecture, Mach is considered the "grandfather of microkernels." Mach services are provided by active objects (Mach tasks) communicating via Mach's IPC primitives. Unfortunately, IPC is rather expensive in Mach. That discourages fine-grained communication, which means Mach active objects tend to be large-grained. The Mach kernel itself is fairly large compared with some of the kernels mentioned previously; it provides virtual memory, a rich IPC primitive, device drivers, and a filesystem [BGJ+92, Loe92, BRS+85]. Tanenbaum's survey compares Mach with the Chorus and Amoeba microkernels [Tan95]. IBM's Workplace OS is a Mach microkernel-based design that is meant to offer multiple operating system personalities simultaneously [Fle97, Raw97]. Another system based on Mach is the GNU Hurd, an operating system whose services are supplied by multiple servers [Fou96].

Snowflake draws on the minimalist theme of microkernels. The projects mentioned above enable functionality to be removed from the supervisor-mode code of an operating system kernel, and provided in user-mode code. That is a necessary first step in allowing users to supply abstractions, rather than requiring administrators to control them. The hierarchical structuring of Flex creates parity between users and administrators. It allows a user to export a subset of his resources using the same interface that an administrator provides, obscuring the distinction (in mechanism) between user and administrator.

8.2 Kernel extension

The opposite strategy to moving abstraction into user-level is to allow users to modify the abstractions implemented in the operating system kernel. Such systems are referred to as "kernel extension" technologies.

VINO is an extensible kernel that allows user-level extensions called "grafts." Pointer safety, that is, protection from errant reads and writes to illegal addresses by untrusted extensions, is provided by software fault isolation. The current work focuses on meaningful protection for abstract resources available via the in-kernel interfaces [SESS96].

The SPIN system allows users to dynamically link extensions into the kernel. Pointer safety is provided by a trusted compiler of a verifiable language (Modula 3). A facility called *guards* protects the kernel against denial of service by misbehaving extensions holding locks or consuming CPU time indefinitely [BCE+95, BSP+95b].

The SPIN group posits that protection should be implemented in software; hence their focus on extensible kernels [BSP+95a]. Druschel disagrees, claiming that extensible kernel research is "entangled in trying to solve the safety problems that extensibility itself introduces" [DPZ97]. Other researchers have felt similarly, and have instead aimed for "static extensibility," which allows unsafe extensions for research purposes, with resulting products linked statically into production kernels [AFG+97]. Statically extensible systems enable systems research, but they do not help achieve user-level abstraction in production environments.

The J-Kernel is an extensible system based on the software protection afforded by the Java Virtual Machine. It extends the memory access safety provided by the Java language, adding a protection model for resources at a higher level of abstraction [HvE98, HCC+97a].

Kernel extension technology has the same goal as microkernel technology: to provide greater flexibility to user applications than is available in conventional monolithic designs. Kernel extension researchers defend their approach as more efficient, because it avoids interprocess communication delays often associated with microkernels. For Snowflake, the approach is less relevant than the objective: to give users more control over abstractions; specifically over resource aggregation.

8.3 User-level abstractions in traditional systems

Another category of systems provide users of traditional, heavyweight kernel systems with greater expressivity, circumventing the user/administrator disparity inherent in such systems.

To support its NFS filesystem, Sun Microsystems extended the concept of inodes in 4.3BSD Unix to vnodes, an object-oriented interface that allows programmers to implement (in-kernel) new Unix filesystems. Portals are filesystems that redirect operations to user-mode processes. Union mounts allow virtual filesystems to be stacked on top of one another to provide more flexible compositions of separate filesystem name trees [MBKQ96, McK95, SP95, PM95]. Because extension occurs in-kernel, it is available to administrators, but not to ordinary users.

Bershad and Pinkerton's Watchdogs provide a user-level hook into the Unix filesystem. User processes can implement directory and file operations, giving users the same expressive power to implement filesystem objects as administrators have always had by being able to extend the kernel [BP88]. The Sprite distributed system offers similar functionality, allowing user-level processes to emulate a file or I/O device by implementing a 'pseudo-device' [WO88].

The previous examples extend the kernel to provide user-level hooks into the filesystem interfaces. Another approach is to use existing hooks, typically debugging facilities, to alter filesystem functionality for any program explicitly interposed upon. UFO allows user-level implementations of filesystems and other kernel interfaces in Solaris. It works by interposing on system calls from target processes using the `/proc` debugging interface [AISS97]. Jones uses Mach's system-call interception facility to build a user-level framework for interposing on the Unix system call interface [Jon93].

Each of these mechanisms shows the desire to let users provide first-class implementations of filesystem interfaces. Snowflake aims to generalize the concept, so that users can provide first-class implementations for any standard system interface.

8.4 Single-system-image clusters

In several operating systems, kernels running on individual hosts connected by a network conspire to present users with a coherent single-system image. These systems are typically configured and maintained by a single administrative entity.

MOS is a multicomputer OS that makes a cluster of homogeneous computers behave as a single-machine UNIX system. Location transparency is implemented using remote procedure calls [BL85].

The Spring distributed system was developed at Sun. Spring nodes run a microkernel called a *nucleus* that implements *doors*, the basic IPC mechanism. Nodes can be coalesced into a *village*, which is the unit of administrative autonomy in Spring [MGH+94, HK93, NR94]. Solaris MC (Multi-Computer) was a Sun effort to integrate Spring concepts into the commercial Solaris operating system [KBM+95].

In Amoeba, nodes communicate in a cluster using custom (usually broadcast) protocols to locate services. Nodes provide computational, storage, and user-interface services to cluster users. An Amoeba cluster communicates with the outside world, including other Amoeba clusters, using IP protocols [TvRvS+90, MvRT+90].

Plan 9 from Bell Labs is a successor to Unix. In Plan 9, most resources are accessed via the filesystem interface; and furthermore, user processes are free to implement the filesystem interface. A typical Plan 9 cluster has user interface terminals, compute nodes, and disk storage nodes. A minimal distributed filesystem protocol makes resources uniformly accessible whether local or remote [PPTT92, PPT+93, PW93, PPD+95].

The Distributed Computing Environment from OSF formally unites several ad-hoc cluster technologies from the Unix world to provide a standard Unix cluster platform [Joh91].

OSF/1-AD2 is a distributed version of OSF's Unix. It virtualizes Unix resources such as vnodes, sockets, and processes to make them location transparent [RNB96].

The Off distributed microkernel project is built on the philosophy that distribution should be provided transparently at a very low level, so that extensions to the operating system do not risk destroying the property of transparent distribution. In Off, "the whole network [cluster] is considered to be the exported and multiplexed hardware" [BF97].

The Sprite network operating system hides distribution by remote file access and cache consistency. The tree of file (and I/O device) names is identical on every machine, and is easy to administrate, as client nodes automatically discover when a file has moved from one server to another [OCD+88]. However, this model assumes a single administrative domain. It derives much administrative convenience from clients' use of broadcast to discover file locations, which does not scale beyond a tightly administrated cluster on a LAN.

Several of the systems described above rely on a facility for transparently hiding distribution across the network cluster. Waldo et. al hold the position that transparent distribution is bad, especially as that distribution scales across (slow and unreliable) world-wide links [WWWK94].

Cristian provides a good overview of fault tolerance, and discusses how it should be applied at the hardware, operating system, and application levels [Cri91].

8.4.1 Single-address-space operating systems

An important category of single-system image clusters are those implemented using a single-address-space operating system (SASOS). A SASOS provides a single-system image at a very low level: each host's CPU is essentially part of a multiprocessor with a large distributed memory. Distributed shared memory (DSM) systems are related, but typically support single applications, rather than providing a cluster-wide single-system image.

Opal exploits the fact that all data can be potentially visible to any domain in a SASOS system to allow programs to share pointer-rich data structures easily. Unfortunately, such sharing is not practical beyond the local network address space [CLFL94].

Angel is a SASOS that uses DSM to give the illusion of a single address space across a cluster. Angel analyzes DSM traffic to determine causally-related process groups, and takes consistent checkpoints of those groups to implement persistence [MWSK94].

Mungi is a single-address-space operating system built on top of L4. It uses capabilities to make protection transparent to applications that do not need to explicitly manipulate capabilities. Device drivers are simply user programs that use capabilities to access the part of virtual memory that maps the device's I/O registers [HEV$^+$98].

Brevix is a commercial single-address-space design for a cluster operating system, targeted at parallel applications. Its naming scheme decouples naming from storage, a characteristic we regard as important for scalable single-system images [FCH$^+$93].

Kotz and Crow examined how SASOS designs depend on hardware with sufficient bits of address space [KC96]. Ansley and Ashton discuss the challenges of administering a SASOS system; specifically the impact of a distributed memory on resiliency in the event of component failure [AA96].

All of the systems described above unify the resources of a cluster of machines to provide a single-system image. In each case, however, there is a cluster administrator responsible for constructing the single-system image. Spring addresses wider distribution by allowing hierarchical administration; however, this still is not "across administrative domains," since hierarchy requires the sub-domains to belong to common administrative superdomain. The vision of Snowflake is to allow users to enjoy a single-system image that knows no administrative boundaries.

8.5 Worldwide systems

The Legion project aims to make a worldwide virtual computer by combining unused workstation resources and exporting them to remote users. It is targeted at exploiting unused parallel processing power. The design philosophy differs from Snowflake in that Legion uses middleware to build a single virtual machine, and then multiplex that resource among interested users [GWtL97, GWF$^+$94, GNTW95, LG96].

GLOBE is a proposal for a middleware-based worldwide operating system. The central concept in GLOBE is a distributed shared-object mechanism that replaces other communications paradigms, such as stream-based synchronous communication (TCP, RPC), shared state with relaxed consistency (NFS, DSM), or remote shared objects (CORBA) [HST96].

The Prospero "Virtual System Model" is a user-centered (more precisely, object-centered) scheme for naming the global set of objects. Prospero objects do not have user-definable interfaces, so a Prospero object cannot be simultaneously a directory and a file [Neu92a, Neu92b]. Snowflake also has per-object namespaces, but its use of explicit interfaces means that any name can refer to an object that has a directory interface, a file interface, and perhaps some other interface.

Microsoft's Millenium project aims to create a seamless, worldwide system. Millenium should appear logically as one global system [BDF+97, Ham98]. Sun's Jini proposal has similar goals, but instead of trying to create a global system, Jini facilitates device discovery in the local area, allowing physically colocated clients and services to discover each other and interoperate [Wal98, Sun98a].

I consider the category of world-wide systems to be Snowflake's closest competitors. Any system that attempts to span the world must clearly address spanning administrative domains; in my work I will address how my solution to this issue differs from the approaches mentioned above.

8.6 Parallel applications and clusters

The goal of many distributed systems is to turn a cluster of systems into a parallel processing workhorse. For this sort of system, the challenge is how to map a single application onto a collection of distributed resources. Tripathi and Karnik provide a useful survey of multiprocessor and distributed system designs [TK95].

Hurricane is a distributed-shared-memory operating system structured using a concept called hierarchical clustering. Hierarchical clustering aims to better scale parallel operating system components by hierarchically subdividing any hotly contended component [UKS95].

PANDA is a runtime package that provides compiled applications with a distributed shared memory environment [ABB+93b].

V and its successor V++ are also distributed systems designed to turn a cluster into a transparent parallel machine. The cache kernel in V++ allows user-level "application kernels" to implement resource management policies. The kernel treats each system resource as a cache for some more plentiful resource, and reflects cache miss events to the user-level application kernels to handle [CK96, GC96, CD95].

OSF/1 AD TNC is a variant of OSF Unix designed for parallel multicomputers. It provides a 'global' (cluster-wide) name space, remote fork, and process migration [ZRB+93].

Disco uses virtual machines to run multiple copies of Irix (a commercial operating system from SGI) on a multiprocessor. The implementation exports a single-system image to users, while providing scalable multiprocessor performance [BDR97b].

Distributed shared memory systems (DSMs) provide applications with a single-system image, where the image is that of a shared-memory region accessed by multiple processes. Shasta, Cashmere-2L, and Peace are software-based distributed shared memory systems [SG97, SDH+97, CGS+97]. ICARE is a software DSM with support for persistence [KMB98]. Per-application DSMs are conceptual forerunners of SASOS designs, which we discussed in Section 8.4.1.

Banerji et. al propose a "substrate object model" for building multiprocessor operating systems. Important goals of their architecture are the creation of a single-system image across the cluster, and support for dynamic evolution of the operating system [BKTC93].

Fox et. al describe a Scalable Network Services architecture that they use to create parallel applications with loose consistency requirements, such as an Internet search engine and a web proxy [FGC+97].

The systems mentioned above generally aim to scale a parallel application to large parallel and distributed systems. In contrast, Snowflake's goal is to use the global large distributed system to run many not necessarily parallel applications. For Snowflake, performance is secondary to accessibility and functionality.

8.7 Distributed language support

One approach to application-at-a-time distribution is to provide distribution support in the language, so that application binaries contain runtime support for distribution.

Chin and Chanson survey distributed object-based programming systems, systems that are the union of autonomous-object-based design and distributed operating systems [CC91].

Distributed Oz is a distributed language that separates computational semantics from distribution semantics. Distributed Oz programs can use write-once variables, which have favorable implementations in a distributed system [vHS97].

Distributed applications are often based on a distributed notification or communications substrate. Lotus uses a protocol called NSTP [Day97]. Reliable multicast is a protocol that provides TCP-like semantics for a group of processes communicating via multicast [WMK95]. Chiueh's distributed virtual world is an example of how application-specific knowledge can be used to optimize distributed communications [Chi97].

8.8 Middleware infrastructure

Middleware is a layer of software that runs on a variety of native operating systems, providing applications with a consistent interface over which to communicate. The Object Management Group's CORBA and Microsoft's DCOM are the current competing standards for object middleware [YD96]. Orfali et. al provide a nice introduction and comparison between the two [OHE96].

CORBA grew out of the Spring operating system's object model, and provides hooks for extensions technologies such as persistence and caching [MGH+94, KAD96]. Isis and its successor Horus are toolkits that encapsulate consistency and concurrency issues, providing distributed applications with a convenient programming model of "virtual synchrony" [BJ87, vBM96].

Prospero, Legion, and GLOBE, which we describe in Section 8.5, are implemented as middleware, to add a new layer of abstraction while being available on a variety of existing systems [Neu92a, GWtL97, HST96].

Metis is a language-independent application framework for "thin clients." User interfaces are transparently distributed to client machines, while application logic runs on back-end servers. Metis' contribution is the virtualization of existing system resources as "services" in the Metis middleware layer [ZPMD97].

Gold Rush is middleware designed to help applications perform well in a mostly-disconnected network environment [BCC+97].

Hadas is a middleware framework that helps application components coordinate to perform distributed computations. Hadas provides mechanisms for smoothing over syntactic and semantic interface mismatches between components [BSCHL97].

8.9 Distributed filesystems

The problem of distributing filesystems is simpler than distributing arbitrary resources, and yet it is by no means a trivial problem. Distributed filesystems provide insight into the challenges of naming in a global environment and transparent distribution across slow and unreliable networks. There are several useful surveys of distributed filesystem technologies that serve as a useful introduction to the topic [LS90, Wel94, DMST95, AEK96].

The Andrew File System is the first distributed filesystem to successfully span administrative domains. It employs aggressive caching to contend with slow networks, and uses a global namespace in which hostnames appear in each path prefix. Naming is coupled to storage location, and AFS configuration depends upon administrators with special privileges [SS94b, MSC+86, GBK+95]. DFS is the distributed filesystem component of the DCE standard; it derives from AFS [Les90].

The World Wide Web can be viewed as a distributed filesystem. URLs provide a global host-based namespace. HTTP defines GET and POST operations which are meant to map to file reads and writes, database selects and updates, or other operations on some lower layer local to the server [BLCL+94]. Because HTTP has a loose consistency model, it can be optimized using several caching techniques [BDR97a, GRC97, MS97].

I mentioned Prospero first in Section 8.5; it is not a distributed filesystem as much as a distributed naming scheme. Storage and access methods are left to the layers below Prospero [Neu92a].

Query routing is a technique for helping distributed resource queries to scale to the very wide area. The idea is to distribute compressed indices for stored data; the indices may indicate false positives for some queries to achieve greater compression. The results of querying the compressed indices gives a list of hints on where to look for the queried data that are verified by trying each in turn [LW97].

Frangipani is a distributed filesystem that is meant to be run within a single administrative domain. It uses a shared virtual disk model [TML97].

The LOCUS filesystem employs file replicas for efficiency; accesses are synchronized through a "centralized synchronization site" [WPE+83, HJ86]. Thus LOCUS filesystems are constrained to a single administrative domain.

8.10 Mobility and disconnection

A worldwide distributed system includes devices that have slow or intermittent connections to the rest of the network. Mobile hosts often have low bandwidth, high latency, and often entirely disconnected network service. To accommodate such hosts in Snowflake, I have to consider how I might apply these results in the Snowflake environment.

Prefetching files ensures that the files are immediately available at some later time, when the network is slow or unavailable. Lei and Duchamp built an online predictor that prefetches files based on previous observed correlations [LD97]. Kuenning and Popek have a similar system that predicts which files a user will need during disconnected operation, and "hoard" them in the mobile host [KP97b]. The Coda distributed filesystem hoards files in a local cache, so that they are available for disconnected operation. Coda allows optimistic file modification while disconnected, and performs mostly-automatic reintegration to resolve write conflicts [KS92].

To evaluate mobile distribution solutions, one needs a reliable way to reproduce the unreliable network service the solution is designed to alleviate. Satyanarayanan and Noble have developed a trace-driven simulation technique that repeatably recreates the mobile network environment in the lab [SN97].

The bandwidth limitations of mobile networks can be addressed by using a proxy server to convert protocols and even degrade content to reduce load on the slow link. Zenel and Duchamp have a general purpose proxy architecture [ZD97]. Fox et. al provide faster web access to mobile clients by providing content-specific compression or fidelity reduction on a well-connected proxy, and forwarding the compressed content over the bandwidth-limited mobile link to the client [FGBA96]. Banga et. al use an optimistic technique to exploit idle link time by prefetching old versions of files that may be updated later with only a small "delta" later [BDR97a].

Noble et. al address the use of variable-bandwidth network connections. Applications indicate their needs, and the system tells the applications how to tune themselves to best use available bandwidth [NSN+97].

Duchamp has created a toolkit that involves users themselves in the process of managing file caching and prefetching on a mobile host [Duc97].

Mobile agents are processes that can migrate between hosts. I have mentioned how file migration can be useful for mobile hosts. Mobile agents are a related concept. A mobile agent moves some amount of computation off the mobile host, where it will have better or more reliable network access. At a later time, the agent can return to the mobile host to report its results. Concordia is an infrastructure that facilitates interaction among mobile agents [WPW+97]. Agent Tcl is part of the D'Agents project at Dartmouth, and allows Tcl "agents" to jump from site to site, carrying their state, including the state of the jumping thread [KGN+97].

8.11 Process migration

Mobile agents are a special case of process migration, which is often used in a single-system environment for load balancing. Here are just a few examples: The LOCUS distributed filesystem supports dynamic load balancing using process migration [HJ86]. Plan 9 and Amoeba have similar functionality. Smith and Ioannidis have implemented a remote `fork()` call with semantics similar to the basic Unix `fork()` [SI88]. Roush's dissertation describes how to minimize the latency associated with process migration [Rou95]. Sprite migrates processes around the cluster, to provide load balancing. Process images are backed to files, which are transferred between systems like any other file [OCD+88]. Snowflake could achieve similar migration between homogeneous architectures by migrating Icee checkpoint images.

8.12 Persistence

Snowflake exploits orthogonal persistence; that is, any object, regardless of type, can be made persistent. In this section, we discuss various approaches to persistence, with a certain bias toward Java persistence, as the Snowflake prototype requires persistence in the context of Java.

The PJama group modified the Java Virtual Machine to add orthogonal persistence. Objects are swizzled out to a very large (possibly larger than a virtual address space) persistent store, and can be reloaded into a different JVM in the future [Spe96, Jor96, MH96, ADJ+96, PAD+97].

IBM's JavaSAS project maps objects into a simulated large address space, then makes that address space shareable and persistent. Persistence is not transparent, because persistent object must be explicitly created with a "factory" method [Mal96, JMN+97].

Another approach to Java object persistence is to push objects out to a relational or object-oriented database. Such techniques generally have the advantage of being implemented entirely in Java, making them portable. Also, they export data to storage that exists outside of the language runtime system, so the data is independent of the operating system or JVM version it was created in. Such systems cannot make all classes persistent however, and so are not type orthogonal. They also typically cannot make threads persistent. Examples of such systems follow.

- Dos Santos and Theroude's system is an example of the relational database mapping approach. Java classes are mapped to a relational database table definition, so that each instance object is represented by a row in the table. They propose to improve the transparency of their system by rewriting bytecodes on the fly [dST96].

- The object-oriented database (OODB) mapping approach is demonstrated in Jspin. Jspin is a Java port of the SPIN system,[2] which maps language record datatypes to an external

[2]This SPIN stands for "Seamless integration of Persistence, Interoperability, and Naming," and is unrelated to the University of Washington SPIN extensible kernel.

object-oriented database [WKMR96, RTW97]. ObjectStore is a commercial product that also implements persistence as an OODB mapping. It requires that all persistence-capable classes derive from the `Persistent` class; therefore, the system is not orthogonally persistent [O'B96, LLOW91].

A significant challenge for persistent object systems is the class evolution problem. To ensure the integrity of persistent object representations, most techniques make class definitions themselves persistent. Without explicit support, one cannot change the class definition in the persistent store, and therefore the data is "trapped" with a certain version of the class that created it. Morrison et. al partially address this problem in their orthogonally persistent PS-Algol and Napier88 languages [MCKM96].

Cooper and Wise offer a valuable critique of orthogonal persistence. Their position is that unrestricted orthogonal persistence sacrifices certain advantages of explicit persistence. The `open/close` interface of traditional filesystems delimit the scope of intended access or modification, which can be useful for caching and prefetching optimizations. They also provide a boundary for access control, and delimit a unit for distribution and copying. Directories in a filesystem *contain* their member files in the sense that any potential user must traverse the directory to reach its contained files [CW96].

Another solution to the lack of transaction boundaries in orthogonally persistent languages is to introduce nested transactions. To achieve orthogonal persistence with transactions, no part of the code may run without being inside at least one transaction. Transactions can also introduce counterintuitive semantics: local variables can roll back when an enclosed transaction aborts. Finally, nested transactions cannot all be ACID,[3] because the durability of the inside transaction is at odds with the atomicity of the enclosing transaction. Nettles et. al explore the advantages and difficulties of transactions in detail [GN96, NW91, BZ98].

Persistence in the Spring system is not transparent; instead, it appears to have an explicit interface much like that of Java's Serializable mechanism [RMP93]. Java has a Serializable interface that indicates to the Java Virtual Machine that a given class may be flattened into a byte stream format and written to an external data stream. Java provides the mechanisms that ensure that only one instance of an object appears in a given stream, so that object retains referential integrity when it is reconstituted. Without referential integrity, two references to the same object in the original state may become different references to two copies of the same object in the reconstituted state [RWWB96].

The Grasshopper operating system is orthogonally persistent. Its creators have proposed running Java on Grasshopper as a way of seamlessly endowing a transient object system with orthogonal persistence [RDH$^+$96, DHF96].

The SHORE object repository combines object-oriented database mapping with transactions to support parallel transaction processing [CDF$^+$94].

The Rio file cache provides an "unstable" form of persistence. The central observation of Rio is that accidental operating system failure is no more likely to corrupt write-protected memory pages than it is to issue bogus writes to disk blocks. If an uninterruptible power supply protects against the most common hardware failures, and the virtual memory system protects against software failures, then Rio can treat main memory as persistent storage, and provide very high performance transactions [LC97].

Frigate is a user-extensible filesystem based on persistent CORBA objects [KP97a].

[3]ACID stands for atomicity (a transaction appears either complete or never initiated), consistency (logical invariants over the database can be maintained), isolation (operations of transactions are not interleaved in time), and durability (once committed, a transaction is persistent).

HAC is a distributed, persistent object store that enjoys the efficiency of a DSM (a page-cached system) while avoiding false sharing by being aware of individual object locations and access patterns [CALM97].

8.12.1 Checkpointing

One approach to implementing transparent persistence is to checkpoint the state of applications that contain some data we wish to make persistent. In this section we discuss how checkpointing works and give some examples of systems that use it; in the next section we explore how to scale checkpointing for distributed systems.

Lee and Anderson provide an excellent overview of fault tolerance [LA90]. Chapter 7 covers error recovery. Backward error recovery is the act of rolling a process back to some prior known state that had been somehow made persistent. The book describes two approaches to providing persistence for the purpose of backward error recovery: state-based error recovery, which involves taking a consistent checkpoint of the entire state of a system; and action-based error recovery, which involves logging the changes to the system, and replaying them to arrive at a known state. The state-based technique introduces latency when taking a snapshot of a system's state, but allows fast recovery because the saved image is basically ready to use. Action-based error recovery introduces very little delay during normal runtime, but replaying the logged changes can introduce substantial delay at recovery time. Bowen and Pradhan discuss several applications of checkpointing, from processor checkpointing (rolling back a few instructions) to redundant memory systems [BP93].

The portable `libckpt` package provides a checkpointing service to generic Unix processes. It is designed to protect long-running scientific computations from system crashes [PBKL95].

My *Icee* checkpointer was modeled on `libckpt`, but with substantial additions to capture more of the Solaris-specific state associated with the target process. Icee can checkpoint and restore a Java Virtual Machine, which makes many more demands of the operating system than a generic scientific program. By making the entire JVM persistent, Icee provides Java with orthogonal, transparent persistence [How98].

Unlike the approaches above, which extract checkpoint state from the runtime process using support from the operating system, the Porch checkpointer rewrites C code to endow it with persistence support. The rewritten code can walk up its own stack to save thread state, and then recreate that stack at runtime. Unlike checkpointers that snapshot the process memory, a program rewritten by Porch knows about the data types it is saving, and can write them to stable storage in a portable format that preserves type information. Thus Porch checkpoint images can be moved to another platform and recovered with a version of the program binary compiled under the new architecture. Porch has difficulty resolving the ambiguities present in C due to `union` datatypes and pointer aliasing [SR98].

Liedtke's L3 is an orthogonally persistent operating system implemented using periodic system-wide checkpoints. Every task in the system (except for device drivers, which must be able to explicitly reconstruct the state of the hardware device they control) is persistent. The system has been used in a real environment, and its users have found the reliability of orthogonal persistence to be very useful [Lie93].

KeyKOS also takes system-wide checkpoints to provide persistence. It uses asynchrony heavily to reduce the impact of checkpointing on system response time, and even to migrate checkpoints to tape to create off-site backups [Lan92].

Lomet and Weikum's checkpointer is designed to checkpoint database applications that run as the second tier in a three-tier system.[4] This application is interesting because databases are

[4]In a three-tier system, the database itself is the first tier, application-specific code run on or near database server

typically persistent and highly available, but database applications, which provide access to the databases, are often not made fault-tolerant [LW98].

8.12.2 Distributed checkpointing

Checkpointing a distributed system efficiently is challenging. A basic checkpoint involves freezing the state of an entire system, copying that state to stable store, then resuming it. A distributed system by its nature is larger than its component nodes; checkpointing a larger system introduces greater delay while the system is frozen. Furthermore, synchronizing a loosely-connected system to ensure that all of its components are stopped simultaneously is slow. Distributed checkpointing algorithms improve this situation by independently checkpointing smaller subsets (typically individual nodes) of a distributed system, then somehow ensuring that some set of node checkpoints together form a coherent checkpoint of the entire system. Strom and Yemini's article is a seminal work on optimistic recovery; that is, allowing checkpointing and computation to proceed asynchronously while logging internode messages to ensure a coherent checkpoint state is reachable [SY85]. Li, Naughton and Plank present a real-time distributed checkpointing algorithm [LNP90].

Distributed checkpointing is a natural extension to a distributed shared virtual memory, to make that DSM persistent. Morin and Puaut survey recoverable DSMs [MP97].

The POGS system provides a "pessimistic" synchronization server for distributed checkpointing in a CORBA environment. It works by coordinating member nodes so that they do not become causally dependent (exchange a message) without sharing a consistent checkpoint first [Zwe97].

An alternative approach taken by the Polar architecture is to simply make checkpoints blindingly fast. By providing redundant hardware to maximize checkpointing I/O performance, Polar allows checkpoints to be taken very rapidly. Because checkpoints are cheap, a node of a distributed computation can checkpoint immediately after every external interaction, thereby avoiding causal dependency or any need for rollback [Chi93].

In summary, there are a wide variety of approaches to persistence. One of the most important distinguishing characteristics of a persistent system is its transparency. Snowflake will provide transparent persistence based on Icee to encourage the use of explicit interfaces based on object-oriented implementations [How98].

8.13 Security

Snowflake will need to address security issues in a unique way. Traditional systems can provide security mechanisms, and assume that a central administrative authority will dictate consistent security policies. Because access spans administrative domains, we must also consider how Snowflake's mechanisms and administrators' policies interact among different administrative domains.

Security is concerned with four key tasks. *Authentication* enables an entity to reliably identify the second party it is communicating with. *Integrity* assures one that his data has not been modified without his knowledge. *Privacy* assures two parties that no others are privy to their communication. *Denial-of-service prevention* allows an entity to share only a bounded fraction of its resources with another.

Java approaches the denial-of-service problem by sandboxing untrusted applications in a safe language environment. Users can share some of their resources with a Java application, knowing that Java will prevent it from taking more resources than expected. Java 1.2 refines the original Java Applet security sandbox with fine-grained access control [GMPS97].

---

is the second tier, and the user interface run on terminals is the third tier.

The J-Kernel also adds fine-grained access control to Java, and adds the ability to revoke capabilities [HCC+97b]. Nagaratnam and Byrne's system adds access control lists to Java. Users can specify in a fine-grained way what resources applets from a given source can access [NB97]. Wallach et. al give an overview of several security extensions to the Java safe language foundation [WBDF97].

Janus provides a sandbox environment for Unix processes by interposing on the process' system calls using the `/proc` debugging interface, and restricting the resources that can be requested from the operating system [GWTB96].

TRON is a Unix kernel extension that provides fine-grained capability-like access control around processes. TRON keeps track of capabilities as they are passed to `fork()`ed child processes [BBS95].

Liedtke et. al propose to sandbox downloaded executable content to prevent denial-of-service using the L4 microkernel. Trusted user-level servers monitor resource consumption [LIJ97].

Legion defers security issues to application programmers, requiring users to enforce security in the classes they design [WWK95]. This suggests that both mechanism and policy are implemented by applications programmers, and neither decision is left to administrators or users.

An Amoeba task authenticates itself to a server using a capability to prove it has a right to access an object [MvRT+90].

The hidden software capabilities model decouples protection from application code, so that protection policies can be updated after an application is written [HMdS96].

Forrest et. al propose that, by analogy with ecological systems, computer systems can be protected by diversity. They point out that software flaws open holes in a huge set of systems that share identical software, and propose diversifying and randomizing software on individual systems to increase their resilience to attack [FSA97].

Grimm and Bershad propose using the ideas of levels of trust and nested categories of objects from mandatory access control as a foundation for security in extensible kernel systems [GB97].

Myers and Liskov present a mandatory access control model for untrusted code. This allows users to share information with an untrusted program, but control how that information is relayed to others. The unique element of their work is that it is decentralized: users can declassify information, rather than some central authority as is traditionally required in mandatory access control systems [ML97].

Thompson's *Reflections on trusting trust* is an eye-opening article about trusted software. Anybody thinking about verifying a Trusted Computing Base, the core components of a system required to implement a security design, must consider whether the very tools that build or examine the TCB have been compromised [Tho84].

8.13.1 Key distribution

Authentication, integrity, and privacy can be achieved using various encryption mechanisms. Encryption schemes depend on a limitless supply of keys to achieve these goals; the schemes are versatile, but depend on communicating parties being able to share keys with authentication, integrity, and privacy: the very attributes encryption is used to provide! Thus, distribution of keys is one of the challenges to useful application of encryption to security.

In secret-key encryption, two parties share the same (secret) key. Kerberos is an example of a hierarchical secret-key management scheme. In Kerberos, users share a secret key with a central Key Distribution Center. Using the secret key, the user and the KDC can be assured of each other's identities, and can establish a session key to provide integrity and privacy [NS78, NT94, BM91].

In public-key encryption, one party holds a private key, and the rest of the world knows the corresponding public key. The private key can be used to authenticate its holder to any party. The public key can be used to send intact, private messages to the private key holder [DH76, RSA78].

Public-key encryption facilitates peer-to-peer key distribution. The Pretty Good Privacy package allows users to build up a "web of trust." You can be sure key X belongs to Sue because your friend Alice told you that you can trust Bob who said you can trust Jim's key, and Jim said that X belongs to Sue. Your certainty is only as sure as the links in the chain from you to the key you are inquiring about, but no hierarchical trust structure is required [Gar95].

Kerberos has recently been extended to employ Pretty Good Privacy's web-of-trust concept to facilitate distributed key management in Kerberos authentication environments [SA95].

8.14 Object-oriented structure

Kiczales and Lamping provide an argument for constructing operating systems using object-oriented structures: object orientation reifies the interfaces and implementations of system components. Each interface or implementation is represented by an explicit object, and *meta-object protocols* can be invoked on the objects to modify them. As application requirements expand, meta-object protocols provide a mechanism by which an application can arrange for the specialization of operating-system interfaces and implementations to meet its specific needs [KL93]. Mendelsohn also argues that OS kernels should support and be built from component software [Men97]. Black and Walpole argue that object-oriented encapsulation allows the replacement of implementation at many levels: not just replacing operating-system policies, but replacing code at the library level as well. They describe the Web as a sort of object-oriented operating system [BW95].

The Spring group claims that object-oriented operating systems are "boring," that building systems using object-oriented methodology is merely an exercise in software engineering. Instead, they point out that the use of object-orientation must contribute some useful feature to the system to be of value [HKN91].

The Clouds distributed system provides a coarse object model. Each Clouds object has its own address space, is persistent, and is transparently distributed across the system. They mention the need for class debugging and evolution in a system with "immutable" classes [DC$^+$90, DAM$^+$91].

The Apertos operating system is an example of a system that exploits meta-object protocols to allow control over operating system components. Every object in Apertos has a meta-object that provides control over the base object [ILY95].

The Choices and μChoices systems are object-oriented frameworks that describe a family of operating systems. They allow the user to statically assemble process management, virtual memory, file system, and networking components, and possibly user-supplied extensions, into a complete kernel [JR91, LTC96]. The Choices group built support for persistent objects in C++ to facilitate their design. Classes are reified as runtime objects, and new class definitions can be dynamically loaded [MCK91].

Grasshopper's disk subsystem has a nice application of component structuring that allows disk configuration to be highly modular [dBDF$^+$94].

Another challenge in operating system design is configuration management. The Forest project builds on Java's package mechanism to create a distributed configuration management tool. The tool allows engineers to build unique versioned packages with a script that describes how to assemble the specific versioned components into the final package [JV97].

Sun Microsystems has developed a custom chip that directly executes Java bytecodes [Sun98b]. But Hölzle and Ungar observe that object-oriented languages do not need special hardware support; in fact, what is good for the goose (C) is good for the gander (OO languages), even a pure, dynamically-typed language like Self [HU95].

Persistent object stores can be very large, and stable storage is slower than main memory. Therefore, it can be difficult to efficiently garbage collect stable object storage. Liskov et. al address this issue in the context of their Thor object database [LMN96].

Baquero et. al discuss how they added concurrency control to BALLOON, a statically-typed, compiled object-oriented language not unlike Java, in a way that avoids conflict with inheritance. Programmers defining subclasses or otherwise reusing existing classes can define new concurrency semantics for the existing class without modifying it [BOM95].

Carroll describes how the active object model reduces how often programmers must explicitly manage critical sections. This allows programmers to exploit concurrency while reducing exposure to difficult concurrency bugs [Car98].

8.14.1 Message passing

One of the challenges of distributed, object-based systems is how to pass around objects and object references. The problems are related to those found in remote procedure call, but RPC generally supports basic datatypes and passing small data structures by copy. Object systems tend to have dense graphs of interconnected objects, making the semantics of remote communication nontrivial.

Smart Messages implement efficient remote method invocation for C++. C++ templates wrap the argument list for a method call into a single C++ object, which ensures that the arguments are contiguous in memory. Then that object can be easily transferred over a byte stream to a remote object server [AOW96].

Java's Remote Method Invocation system uses Java Serialization to transfer objects by copy, and provides a stub and skeleton compiler to create proxy objects that represent objects transferred by reference. Programmers decide at class-definition time how an object is transferred, by copy or by reference, by marking their class as implementing the Java interface `Remote` [WRW96]. The RMI group has also considered how to provide automatic activation of remote objects, so that resources can be freed by putting unused objects "to sleep," knowing that they will be transparently awakened when they are referenced by a client [WWW95].

Objects can be passive or active. When a client calls a method in a passive object, the client donates its thread for the object to use while executing the method's code. Active objects have one or more internal threads; each thread waits on a queue for an incoming message, then executes the method associated with that message. Ford and Lepreau argue that while passive and active object models are equivalent, it makes more sense to natively support passive objects, and implement active objects with a passive object and a thread [FL93].

Waldo et. al describe an event model for use in a system with remote method invocation. In a distributed system, components and classes can evolve, so their events have no universal namespace; instead, an event's name is only meaningful between its sender and its receiver [WWWK95].

The remote procedure call system used in the Nexus distributed system improves on completely static stub/skeleton designs such as Sun RPC, DCE, and CORBA, by using runtime checks to allow clients and servers of a given interface to evolve independently [TN98].

As a system designed to scale across administrative domains, Snowflake must draw on a wide variety of operating-systems research to address concerns from naming to security. The previous work described above has informed and will inform our future direction for developing Snowflake.

Acknowledgements

Thanks to my advisor David Kotz for guidance in the project and his always thoughtful comments on writing. We wish to gratefully acknowledge the support of the USENIX organization for supplying funding for the author to develop Snowflake. Thanks also to Sun Microsystems for providing software used in this project.

References

[AA96] Carl Ansley and Paul Ashton. System administration in a single address space operating system. In *Proceedings of the Fifth International Workshop on Object Orientation in Operating Systems*, pages 166–170, 1996.

[ABB+93a] H. Assenmacher, T. Breitbach, P. Buhler, V. Hübsch, and R. Schwarz. The PANDA system architecture: A pico-kernel approach. In *Proceedings of the Fourth Workshop on Future Trends of Distributed Computing Systems*, pages 470–476, September 1993.

[ABB+93b] Holger Assenmacher, Thomas Breitbach, Peter Buhler, Volker Hübsch, and Reinhard Schwarz. PANDA — supporting distributed programming in C++. In O. Nierstras, editor, *Proceedings of 1993 European Conference on Object-Oriented Programming*, volume 707 of *Lecture Notes in Computer Science*, pages 361–383, Kaiserslautern, Germany, July 1993. Springer-Verlag.

[ABB+94] H. Assenmacher, T. Breitbach, P. Buhler, V. Hübsch, H. Peine, and R. Schwarz. Meeting the application in user space. In *Proceedings of the Sixth ACM SIGOPS European Workshop*, pages 82–87, September 1994.

[ADJ+96] Malcom Atkinson, Laurent Daynès, Mick Jordan, Tony Printezis, and Susan Spence. An orthogonally persistent Java. *ACM SIGMOD Record*, 25(4):68–75, December 1996.

[AEK96] Ahmed Amer and Amr El-Kadi. Beating bottlenecks in the design of distributed file systems. *;login: the USENIX Association Newsletter*, 21(6):14–23, December 1996.

[AFG+97] M. Auslander, H. Franke, B. Gamsa, O. Krieger, and M. Stumm. Customization lite. In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 43–48, 1997.

[AISS97] Albert D. Alexandrov, Maximilian Ibel, Klaus E. Schauser, and Chris Scheiman. A novel way of extending the operating system at the user level: the UFO global file system. *;login: the USENIX Association Newsletter*, 22(2):40–41, April 1997.

[AOW96] Eshrat Arjomandi, William G. O'Farrell, and Gregory V. Wilson. Smart Messages: An object-oriented communication mechanism for parallel systems. *Computing Systems*, 9(4):313–329, Fall 1996.

[BBS95] Andrew Berman, Virgil Bourassa, and Erik Selberg. TRON: Process-specific file protection for the UNIX operating system. In *Proceedings of the 1995 USENIX Technical Conference*, pages 165–175, January 1995.

[BCC+97] M.A. Butrico, H. Chang, A. Cocchi, N.H. Cohen, D.G. Shea, and S.E. Smith. Gold Rush: mobile transaction middleware with Java-object replication. In *Proceedings of the Third USENIX Conference on Object-Oriented Technologies*, pages 91–101, 1997.

[BCE+95] Brian N. Bershad, Craig Chambers, Susan Eggers, Chris Maeda, Dylan McNamee, Przemysław Pardyak, Stefan Savage, and Emin Gün Sirer. SPIN: An extensible microkernel for application-specific operating system services. *ACM Operating Systems Review*, 29(1):74–77, January 1995.

[BDF+97] W.J. Bolosky, R.P. Draves, R.P. Fitzgerald, C.W. Fraser, M.B. Jones, T.B. Knoblock, and R. Rashid. Operating system directions for the next millennium. In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 106–110, 1997.

[BDR97a] Gaurav Banga, Fred Douglis, and Michael Rabinovich. Optimistic deltas for WWW latency reduction. In *Proceedings of the 1997 USENIX Technical Conference*, pages 289–303, 1997.

[BDR97b] E. Bugnion, S. Devine, and M. Rosenblum. Disco: running commodity operating systems on scalable multiprocessors. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, pages 143–156, October 1997.

[Ber92] B.N. Bershad. The increasing irrelevance of IPC performance for microkernel-based operating systems. In *Proceedings of the USENIX Workshop on Micro-Kernels and Other Kernel Architectures*, pages 205–211, 1992.

[BF97] F.J. Ballesteros and L.L. Fernandez. The network hardware is the operating system. In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 32–36, 1997.

[BGJ+92] D.L. Black, D.B. Golub, D.P. Julin, R.F. Rashid, R.P. Draves, R.W. Dean, A. Forin, J. Barrera, H. Tokuda, G. Malan, and D. Bohman. Microkernel operating system architecture and mach. In *Proceedings of the USENIX Workshop on Micro-Kernels and Other Kernel Architectures*, pages 11–31, 1992.

[BJ87] Kenneth P. Birman and Thomas A. Joseph. Exploiting virtual synchrony in distributed systems. In *Proceedings of the Eleventh ACM Symposium on Operating Systems Principles*, pages 123–138, Austin, TX, 1987. ACM Press.

[BKTC93] A. Banerji, D. Kulkarni, J. Tracey, and D. Cohn. The substrate object model and architecture. In *Proceedings of the Second IEEE International Symposium on High Performance Distributed Computing*, pages 31–40, December 1993.

[BL85] A. Barak and A. Litman. MOS: a multicomputer distributed operating system. *Software— Practice and Experience*, 15(8):725–737, August 1985.

[BLCL+94] T. Berners-Lee, R. Cailliau, A. Luotonen, H. Frystyk Nielsen, and A. Secret. The world-wide web. *Communications of the ACM*, 37(8):76–82, August 1994.

[BM91] S. M. Bellovin and M. Merritt. Limitations of the Kerberos authentication system. In *Proceedings of the 1991 Winter USENIX Conference*, pages 119–132, 1991.

[BOM95] C. Baquero, R. Oliveira, and F. Moura. Integration of concurrency control in a language with subtyping and subclassing. In *Proceedings of the Third USENIX Conference on Object-Oriented Technologies*, pages 173–183, 1995.

[BP88] B.N. Bershad and C.B. Pinkerton. Watchdogs — extending the unix file system. *Computing Systems*, 1(2):169–188, Spring 1988.

[BP93] N. S. Bowen and D. K. Pradhan. Processor- and memory-based checkpoint and rollback recovery. *IEEE Computer*, 26(2):22–31, February 1993.

[BRS+85] Robert Baron, Richard Rashid, Ellen Siegel, Avadis Tevanian, and Michael Young. Mach-1: an operating environment for large-scale multiprocessor applications. *IEEE Software*, 2(4):65–67, July 1985.

[BSCHL97] I. Ben-Shaul, A. Cohen, O. Holder, and B. Lavva. Hadas: a network-centric framework for interoperability programming. *International Journal of Cooperative Information Systems*, 6(3-4):293–314, Winter 1997.

[BSP+95a] Brian Bershad, Stefan Savage, Przemysław Pardyak, David Becker, Marc Fiuczynski, and Emin Gün Sirer. Protection is a software issue. In *Proceedings of the Fifth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 62–65, May 1995.

[BSP+95b] Brian Bershad, Stefan Savage, Przemysław Pardyak, Emin Gün Sirer, Marc E. Fiuczynski, David Becker, Craig Chambers, and Susan Eggers. Extensibility, safety and performance in the SPIN operating system. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 267–284, Copper Mountain, CO, December 1995. ACM Press.

[BW95] Andrew P. Black and Jonathan Walpole. Objects to the rescue! *ACM Operating Systems Review*, 29(1):74–77, January 1995.

[BZ98] Stephen M. Blackburn and John N. Zigman. Concurrency — the fly in the ointment? In *Proceedings of the Third International Workshop on Persistence and Java*, Tiburon, CA, September 1998. Available at: `http://www.sunlabs.com/research/forest/com.sun.labs.pjw3.main.html`.

[CALM97] M. Castro, A. Adya, B. Liskov, and A.C. Myers. HAC: hybrid adaptive caching for distributed storage systems. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, pages 102–115, October 1997.

[Car98] Martin Carroll. Active objects made easy. *Software— Practice and Experience*, 28(1):1–21, January 1998.

[CC91] R.S. Chin and S.T. Chanson. Distributed object-based programming systems. *ACM Computing Surveys*, 23(1):5–48, March 1991.

[CD95] David R. Cheriton and Kenneth J. Duda. A caching model of operating system kernel functionality. *ACM Operating Systems Review*, 29(1):83–86, January 1995.

[CDF+94] M.J. Carey, D.J. Dewitt, M.J. Franklin, N.E. Hall, M.L. McAuliffe, J.F. Naughton, D.T. Schuh, M.H. Solomon, C.K. Tan, O.G. Tsatalos, S.J. White, and M.J. Zwilling. Shoring up persistent applications. *SIGMOD Record*, 23(2):383–394, June 1994.

[CGS+97] Jörg Cordsen, Thomas Garnatz, Michael Sander, Anne Gerischer, Marco Dimas Gubitoso, Ute Haack, and Wolfgang Schröder-Preikschat. Vote for Peace: Implementation and performance of a parallel operating system. *IEEE Concurrency*, 5(2):16–27, April–June 1997.

[Chi93] Tzi-Cker Chiueh. Polar: a storage architecture for fast checkpointing. *Journal of Information Science and Engineering*, 9(1):61–80, March 1993.

[Chi97] Tzi-Cker Chiueh. Distributed systems support for networked games. In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 99–9104, 1997.

[CK96] David R. Cheriton and Robert A. Kutter. Optimized memory-based messaging: Leveraging the memory system for high-performance communication. *Computing Systems*, 9(3):179–215, Summer 1996.

[CLFL94] Jeffrey S. Chase, Henry M. Levy, Michael J. Feeley, and Edward D. Lazowska. Sharing and protection in a single address space operating system. *ACM Transactions on Computer Systems*, pages 271–307, November 1994.

[Cri91] Flaviu Cristian. Understanding fault-tolerant distributed systems. *Communications of the ACM*, 34(2):56–78, February 1991.

[CW96] Tim Cooper and Michael Wise. Critique of orthogonal persistence. In *Proceedings of the Fifth International Workshop on Object Orientation in Operating Systems*, pages 122–126, 1996.

[DAM+91] Partha Dasgupta, R. Ananthanarayanan, Sathis Menon, Ajay Mohindra, and Raymond Chen. Distributed programming with objects and threads in the Clouds system. *Computing Systems*, 4(3):243–275, Summer 1991.

[Day97] M. Day. What synchronous groupware needs: notification services. In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 118–122, 1997.

[dBDF+94] Rex di Bona, Alan Dearle, James Farrow, Frans Henskens, Anders Lindstrom, John Rosenberg, and Francis Vaughan. Generic interface for configurable disk I/O systems. In Gopal Gupta, editor, *Proceedings of the Seventeenth Annual Computer Science Conference*, pages 355–362, Christchurch, New Zealand, 1994.

[DC+90] P. Dasgupta, R. C. Chen, , S. Menon, M. P. Pearson, R. Ananthanarayanan, U. Ramachandran, M. Ahamad, R. J. Leblanc, W. F. Appelbe, J. M. Bernabéu-Aubán, P. W. Hutto, M. Y. A. Khalidi, and C. J. Wilkenloh. The design and implementation of the Clouds distributed operating system. *Computing Systems*, 3(1):11–46, Winter 1990.

[DdBF+94] Alan Dearle, Rex di Bona, James Farrow, Frans Henskens, Anders Lindstrom, John Rosenberg, and Francis Vaughan. Grasshopper: An orthogonally persistent operating system. *Computing Systems*, 7(3):289–312, Summer 1994.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[DHF96] Alan Dearle, David Hulse, and Alex Farkas. Persistent operating system support for Java. In *Proceedings of the First International Workshop on Persistence and Java*, September 1996.

[DMST95] Murthy Devarakonda, Ajay Mohindra, Jill Simoneaux, and William H. Tetzlaff. Evaluation of design alternatives for a cluster file system. In *Proceedings of the 1995 USENIX Technical Conference*, pages 35–46, January 1995.

[DPZ97] P. Druschel, V.S. Pai, and W. Zwaenepoel. Extensible kernels are leading OS research astray. In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 38–42, 1997.

[dST96] C. Souza dos Santos and E. Theroude. Persistent Java. In *Proceedings of the First International Workshop on Persistence and Java*, September 1996.

[Duc97] Dan Duchamp. A toolkit approach to partially connected operation. In *Proceedings of the 1997 USENIX Technical Conference*, pages 305–318, 1997.

[EK95] Dawson R. Engler and M. Frans Kaashoek. Exterminate all operating system abstractions. In *Proceedings of the Fifth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 78–83, 1995.

[FCH+93] Martin Fouts, Tim Connors, Steve Hoyle, Bart Sears, Tim Sullivan, and John Wilkes. Brevix design 1.01. Technical Report HPL-OSR-93-22, Operating Systems Research Department, Hewlett-Packard Laboratories, April 1993.

[FGBA96] Armando Fox, Steven D. Gribble, Eric A. Brewer, and Elan Amir. Adapting to network and client variability via on-demand dynamic distillation. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 160–170, Cambridge, MA, October 1996.

[FGC+97] A. Fox, S.D. Gribble, Y. Chawathe, E.A. Brewer, and P. Gauthier. Cluster-based scalable network services. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, pages 78–91, October 1997.

[FHL+96] Bryan Ford, Mike Hibler, Jay Lepreau, Patrick Tullmann, Godmar Back, and Stephen Clawson. Microkernels meet recursive virtual machines. In *Proceedings of the 1996 Symposium on Operating Systems Design and Implementation*, pages 137–151, October 1996.

[FL93] Bryan Ford and Jay Lepreau. Microkernels should support passive objects. In *Proceedings of the Third International Workshop on Object Orientation in Operating Systems*, pages 226–229, December 1993.

[Fle97] B.D. Fleisch. The failure of personalities to generalize. In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 8–13, 1997.

[Fou96] Free Software Foundation. Towards a new strategy of OS design, 1996. Available at: `http://www.gnu.ai.mit.edu/software/hurd/hurd.html`.

[FS96] Bryan Ford and Sai Susarla. CPU inheritance scheduling. In *Proceedings of the 1996 Symposium on Operating Systems Design and Implementation*, pages 91–105, October 1996.

[FSA97] S. Forrest, A. Somayaji, and D.H. Ackley. Building diverse computer systems. In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 67–72, 1997.

[Gar95] Simson Garfinkel. *PGP: Pretty Good Privacy.* O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, 1995.

[GB97] R. Grimm and B.N. Bershad. Security for extensible systems. In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 62–66, 1997.

[GBK+95] Jonathan S. Goldick, Kathy Benninger, Christopher Kirby, Christopher Maher, and Bill Zumach. Multi-resident AFS: An adventure in mass storage. In *Proceedings of the 1995 USENIX Technical Conference*, pages 47–58, January 1995.

[GC96] Michael Greenwald and David Cheriton. The synergy between non-blocking synchronization and operating system structure. In *Proceedings of the 1996 Symposium on Operating Systems Design and Implementation*, pages 123–136, October 1996.

[GMPS97] Li Gong, Marianne Mueller, Hemma Prafullchandra, and Roland Schemers. Going beyond the sandbox: An overview of the new security architecture in the Java Development Kit 1.2. In *Proceedings of the First USENIX Symposium on Internet Technologies and Systems*, pages 103–112, December 1997.

[GN96] Alex Garthwaite and Scott Nettles. Transactions for Java. In *Proceedings of the First International Workshop on Persistence and Java*, September 1996.

[GNTW95] Andrew S. Grimshaw, Anh Nguyen-Tuong, and William A. Wulf. Campus-wide computing: Early results using Legion at the University of Virginia. Technical Report CS-95-19, University of Virginia, March 1995.

[GRC97] S. Gadde, M. Rabinovich, and J. Chase. Reduce, reuse, recycle: an approach to building large Internet caches. In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 93–98, 1997.

[GWF+94] Andrew S. Grimshaw, William A. Wulf, James C. French, Alfred C. Weaver, and Paul F. Reynolds Jr. A synopsis of the Legion project. Technical Report CS-94-20, University of Virginia, June 1994.

[GWTB96] Ian Goldberg, David Wagner, Randi Thomas, and Eric A. Brewer. A secure environment for untrusted helper applications: Confining the wily hacker. In *Proceedings of the Sixth USENIX Security Symposium*, pages 1–13. USENIX Association, July 1996.

[GWtL97] Andrew S. Grimshaw, Wm. A. Wulf, and the Legion Team. The Legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1):39–45, January 1997.

[Ham98] Scott Hamilton. Inside Microsoft Research. *Computer*, 31(1):51–58, January 1998.

[HCC+97a] Chris Hawblitzel, Chi-Chao Chang, Grzegorz Czajkowski, Deyu Hu, and Thorsten von Eicken. Implementing multiple protection domains in Java. Technical Report TR97-1660, Cornell University Department of Computer Science, December 1997.

[HCC+97b] Chris Hawblitzel, Chi-Chao Chang, Grzegorz Czajkowski, Deyu Hu, and Thorsten von Eicken. SLK: A capability system based on safe language technology. Department of Computer Science, Cornell University. Internal Memo, March 1997. Available at: `http://www.cs.cornell.edu/SLK/papers.html`.

[HEV+98] G. Heiser, K. Elphinstone, J. Vochteloo, S. Russell, and J. Liedtke. The Mungi single-address-space operating system. *Software— Practice and Experience*, 28(9):901–928, July 1998.

[HHL+97] H. Hartig, M. Hohmuth, J. Liedtke, S. Schonberg, and J. Wolter. The performance of μ-kernel-based systems. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, pages 66–77, October 1997.

[HJ86] A. Hac and T.J. Johnson. A study of dynamic load balancing in a distributed system. *Computer Communication Review*, 16(3):348–356, August 1986.

[HK93] Graham Hamilton and Panos Kougiouris. The Spring nucleus: A microkernel for objects. In *Proceedings of the 1993 Summer USENIX Technical Conference*, pages 147–159, 1993.

[HKN91] G. Hamilton, Y.A. Khalidi, and M.N. Nelson. Why object oriented operating systems are boring. In *Proceedings of the 1991 International Workshop on Object Orientation in Operating Systems*, pages 118–119, October 1991.

[HMdS96] D. Hagimont, J. Mossiere, X. Rousset de Pina, and F. Saunier. Hidden software capabilities. In *Proceedings of the Sixteenth International Conference on Distributed Computer Systems*, pages 282–289. IEEE Computer Society Press, May 1996.

[How98] Jon Howell. Straightforward Java persistence through checkpointing. In *Proceedings of the Third International Workshop on Persistence and Java*, Tiburon, CA, September 1998. Available at: `http://www.sunlabs.com/research/forest/com.sun.labs.pjw3.main.html`.

[HST96] Philip Homburg, Maarten van Steen, and Andrew S. Tanenbaum. Communication in GLOBE: an object-based worldwide operating system. In *Proceedings of the Fifth International Workshop on Object Orientation in Operating Systems*, pages 43–47, October 1996.

[HU95] Urs Hölzle and David Ungar. Do object-oriented languages need special hardware support? In *Proceedings of the Ninth European Conference on Object-Oriented Programming*, pages 283–302, August 1995.

[HvE98] Chris Hawblitzel and Thorsten von Eicken. A case for language-based protection. Technical Report TR98-1670, Cornell University Department of Computer Science, March 1998.

[ILY95] J. I. Itoh, R. Lea, and Y. Yokote. Using meta-objects to support optimisation in the Apertos operating system. In *Proceedings of the Third USENIX Conference on Object-Oriented Technologies*, pages 147–157, 1995.

[JMN+97] Maynard P. Johnson, Steven J. Munroe, John G. Nistler, James W. Stopyro, and Ashok Malhotra. Java(tm) persistence via persistent virtual storage. In *Proceedings of the Second International Workshop on Persistence and Java*, August 1997.

[Joh91] Brad Curtis Johnson. Distributed computing environment framework. Technical Report DEV-DCE-TP6-1, Open Software Foundation, June 1991.

[Jon93] Michael B. Jones. Interposition agents: Transparently interposing user code at the system interface. *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*, pages 80–93, December 1993.

[Jor96] Mick Jordan. Early experiences with persistent Java. In *Proceedings of the First International Workshop on Persistence and Java*, September 1996. Available at: `http://www.sunlabs.com/research/forest/UK.Ac.Gla.Dcs.PJW1.pj1.html`.

[JR91] Ralph E. Johnson and Vincent F. Russo. Reusing object-oriented designs. Technical Report UIUCDCS 91-1696, University of Illinois, 1991.

[JV97] Mick Jordan and Michael L. Van De Vanter. Modular system building with Java(tm) packages. In *Proceedings of the Eighth Conference on Software Engineering Environments*, pages 155–163, April 1997.

[KAD96] R. Kordale, M. Ahamad, and M. Devarakonda. Object caching in a CORBA compliant system. *Computing Systems*, 9(4):377–404, Fall 1996.

[KBM+95] Yousef A. Khalidi, Jose M. Bérnabéu, Vlada Matena, Ken Shirriff, and Moti Thadani. Solaris MC: A multi-computer OS. Technical Report TR-95-48, Sun Microsystems Laboratories, November 1995.

[KC96] David Kotz and Preston Crow. The expected lifetime of single-address-space operating systems. *Computing Systems*, 9(3):155–178, Summer 1996.

[KEG+97] M.F. Kaashoek, D.R. Engler, G.R. Ganger, H.M. Briceno, R. Hunt, D. Mazieres, T. Pinckney, R. Grimm, J. Jannotti, and K. Mackenzie. Application performance and flexibility on exokernel systems. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, pages 52–65, October 1997.

[KGN+97] David Kotz, Robert Gray, Saurab Nog, Daniela Rus, Sumit Chawla, and George Cybenko. Agent Tcl: Targeting the needs of mobile computers. *IEEE Internet Computing*, 1(4):58–67, July/August 1997.

[KL93] Gregor Kiczales and John Lamping. Operating systems: Why object oriented? In *Proceedings of the Third International Workshop on Object Orientation in Operating Systems*, pages 25–30, 1993.

[KMB98] A.-M. Kermarrec, C. Morin, and M. Banâtre. Design, implementation and evaluation of ICARE: An efficient recoverable DSM. *Software— Practice and Experience*, 28(9):981–1010, July 1998.

[KP97a] T.H. Kim and G.J. Popek. Frigate: an object-oriented file system for ordinary users. In *Proceedings of the Third USENIX Conference on Object-Oriented Technologies*, pages 115–129, 1997.

[KP97b] G.H. Kuenning and G.J. Popek. Automated hoarding for mobile computers. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, pages 264–275, October 1997.

[KS92] James J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.

[LA90] P. A. Lee and T. Anderson. *Fault Tolerance: Principles and Practice*, volume 3 of *Dependable Computing and Fault-Tolerant Systems.* Springer Verlag, second revised edition, 1990.

[Lan92] C.R. Landau. The checkpoint mechanism in KeyKOS. In *Proceedings of the Second International Workshop on Object Orientation in Operating Systems*, pages 86–91, 1992.

[LC97] D.E. Lowell and P.M. Chen. Free transactions with Rio Vista. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, pages 92–101, October 1997.

[LD97] Hui Lei and Dan Duchamp. An analytical approach to file prefetching. In *Proceedings of the 1997 USENIX Technical Conference*, pages 275–288, 1997.

[Les90] Norbert Leser. Towards a worldwide distributed file system. Technical Report DEV-DCE-TP4-1, Open Software Foundation, September 1990.

[LES+97] J. Liedtke, K. Elphinstone, S. Schonberg, H. Hartig, G. Heiser, N. Islam, and T. Jaeger. Achieved IPC performance (still the foundation for extensibility). In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 28–31, 1997.

[LG96] M. Lewis and A. Grimshaw. The core Legion object model. In *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, pages 551–561, August 1996.

[Lie93] Jochen Liedtke. A persistent system in real use: Experiences of the first 13 years. In *Proceedings of the Third International Workshop on Object Orientation in Operating Systems*, pages 2–11, 1993.

[Lie95] Jochen Liedtke. On micro-kernel construction. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 237–250, Copper Mountain, CO, December 1995. ACM Press.

[LIJ97] J. Liedtke, N. Islam, and T. Jaeger. Preventing denial-of-service attacks on a μ-kernel for WebOSes. In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 73–79, 1997.

[LLOW91] Charles Lamb, Gordon Landis, Jack Orenstein, and Dan Weinreb. The ObjectStore database system. *Communications of the ACM*, 34(10):50–63, October 1991.

[LMN96] Barbara Liskov, Umesh Maheshwari, and Tony Ng. Partitioned garbage collection of a large stable heap. In *Proceedings of the Fifth International Workshop on Object Orientation in Operating Systems*, pages 117–121, October 1996.

[LNP90] Kai Li, Jeffrey F. Naughton, and James S. Plank. Real-time, concurrent checkpoint for parallel programs. In *Proceedings of the Second Annual ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 79–88, Seattle, WA (USA), March 1990. ACM.

[Loe92] K. Loepere. Mach 3 kernel principles. Technical Report MK67, Open Software Foundation and Carnegie Mellon University, 1992.

[LRD95] Anders Lindström, John Rosenberg, and Alan Dearle. The grand unified theory of address spaces. In *Proceedings of the Fifth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 66–71, May 1995.

[LS90] Eliezer Levy and Abraham Silberschatz. Distributed file systems: Concepts and examples. *ACM Computing Surveys*, 22(4):321–374, December 1990.

[LTC96] Willy S. Liao, See-Mong Tan, and Roy H. Campbell. Fine-grained, dynamic user customization of operating systems. In *Proceedings of the Fifth International Workshop on Object Orientation in Operating Systems*, pages 62–66, October 1996.

[LW97] P.J. Leach and C. Weider. Query routing: applying systems thinking to Internet search. In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 82–86, 1997.

[LW98] David B. Lomet and Gerhard Weikum. Efficient transparent application recovery in client-server information systems. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 460–471, June 1998.

[Mal96] Ashok Malhotra. Persistent Java objects: A proposal. In *Proceedings of the First International Workshop on Persistence and Java*, September 1996.

[MBKQ96] Marshall Kirk McKusick, Keith Bostic, Michael J. Karels, and John S. Quarterman. *The design and implementation of the 4.4BSD operating system*. Addison Wesley, 1996.

[MCK91] Peter W. Madany, Roy H. Campbell, and Panos Kougiouris. Experiences building an object-oriented system in C++. Technical Report UIUC-DCS-R-91-1671, The University of Illinois at Urbana-Champaign, March 1991.

[McK95] Marshall Kirk McKusick. The virtual filesystem interface in 4.4BSD. *Computing Systems*, 8(1):3–25, Winter 1995.

[MCKM96] Ron Morrison, Richard Connor, Graham Kirby, and David Munro. Can Java persist? In *Proceedings of the First International Workshop on Persistence and Java*, September 1996. Available at: `http://www.sunlabs.com/research/forest/UK.Ac.Gla.Dcs.PJW1.pj1.html`.

[Men97] N. Mendelsohn. Operating systems for component software environments. In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 49–54, 1997.

[MGH+94] J.G. Mitchell, J.J. Gibbons, G. Hamilton, P.B. Kessler, Y.A. Khalidi, P. Kougiouris, P.W. Madany, M.N. Nelson, M.L. Powell, and S.R. Radia. An overview of the Spring system. In *Proceedings of COMPCON '94*, pages 122–131, 1994.

[MH96] J. Eliot B. Moss and Tony L. Hosking. Approaches to adding persistence to Java. In *Proceedings of the First International Workshop on Persistence and Java*, September 1996. Available at: `http://www.sunlabs.com/research/forest/UK.Ac.Gla.Dcs.PJW1.pj1.html`.

[MK97] D. Mazieres and M.F. Kaashoek. Secure applications need flexible operating systems. In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 56–61, 1997.

[ML97] A.C. Myers and B. Liskov. A decentralized model for information flow control. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, pages 129–142, October 1997.

[MP97] C. Morin and I. Puaut. A survey of recoverable distributed shared virtual memory systems. *IEEE Transactions on Parallel and Distributed Systems*, 8(9):959–969, September 1997.

[MS97] Stephen Manley and Margo Seltzer. Web facts and fantasy. In *Proceedings of the First USENIX Symposium on Internet Technologies and Systems*, pages 125–133, December 1997.

[MSC+86] James H. Morris, Mahadev Satyanarayanan, Michael H. Conner, John H. Howard, David S. H. Rosenthal, and F. Donelson Smith. Andrew: A distributed personal computing environment. *Communications of the ACM*, 29(3):184–201, March 1986.

[MvRT+90] Sape J. Mullender, Guido van Rossum, Andrew S. Tanenbaum, Robbert van Renesse, and Hans van Staveren. Amoeba: A distributed operating system for the 1990s. *IEEE Computer*, 23(5):44–53, May 1990.

[MWSK94] K. Murray, T. Wilkinson, T. Stiemerling, and P. Kelly. Angel: resource unification in a 64-bit microkernel. In *Proceedings of the Twenty-Seventh Annual Hawaii International Conference on System Sciences*, pages 106–115, January 1994.

[NB97] N. Nagaratnam and S.B. Byrne. Resource access control for an Internet user agent. In *Proceedings of the Third USENIX Conference on Object-Oriented Technologies*, pages 199–208, 1997.

[Neu92a] B. Clifford Neuman. The Prospero file system. In *Proceedings of the USENIX File Systems Workshop*, pages 13–28, May 1992.

[Neu92b] B. Clifford Neuman. The Prospero file system: A global file system based on the virtual system model. *Computing Systems*, 5(4):407–432, Fall 1992.

[NR94] M.N. Nelson and S.R. Radia. A uniform name service for Spring's Unix environment. In *Proceedings of the 1994 Winter USENIX Technical Conference*, pages 201–209, January 1994.

[NS78] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[NSN+97] B.D. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn, and K.R. Walker. Agile application-aware adaptation for mobility. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, pages 276–287, October 1997.

[NT94] B. Clifford Neuman and Theodore Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications*, 32(9):33–38, September 1994.

[NW91] S.M. Nettles and J.M. Wing. Persistence+undoability=transactions. In *Proceedings of the Twenty-Fifth Annual Hawaii International Conference on System Sciences*, pages 832–843, January 1991.

[O'B96] Patrick O'Brien. Java data management using ObjectStore and PSE. Object Design, Inc. white paper, November 1996. Available at: `http://www.odi.com/content/white_papers/javawp1.html`.

[OCD+88] J.K. Ousterhout, A.R. Cherenson, F. Douglis, M.N. Nelson, and B.B. Welch. The Sprite network operating system. *IEEE Computer*, 21(2):23–36, February 1988.

[OHE96] Robert Orfali, Dan Harkey, and Jeri Edwards. *The essential distributed objects survival guide*. John Wiley & Sons, 1996.

[PAD+97] Tony Printezis, Malcolm Atkinson, Laurent Daynès, Susan Spence, and Pete Bailey. The design of a new persistent object store for PJama. In *Proceedings of the Second International Workshop on Persistence and Java*, August 1997. Available at: `http://www.sunlabs.com/research/forest/COM.Sun.Labs.Forest.PJava.PJW2.pjw2.html`.

[PB95] Dave Probert and John Bruno. Implementing operating systems without kernels. Technical Report TRCS95-24, University of California at Santa Barbara, December 1995.

[PBKL95] James S. Plank, Micah Beck, Gerry Kingsley, and Kai Li. Libckpt: Transparent checkpointing under unix. In *Proceedings of the 1995 USENIX Technical Conference*, pages 213–233, 1995.

[PM95] Jan-Simon Pendry and Marshall Kirk McKusick. Union mounts in 4.4BSD-Lite. In *Proceedings of the 1995 USENIX Technical Conference*, pages 25–33, January 1995.

[PPD+95] Rob Pike, Dave Presotto, Sean Dorward, Bob Flandrena, Ken Thompson, Howard Trickey, and Phil Winterbottom. Plan 9 from Bell Labs. *Computing Systems*, 8(3):221–254, Summer 1995.

[PPT+93] Rob Pike, Dave Presotto, Ken Thompson, Howard Trickey, and Phil Winterbottom. The use of name spaces in Plan 9. *ACM Operating Systems Review*, 27(2):72–76, April 1993.

[PPTT92] Dave Presotto, Rob Pike, Ken Thompson, and Howard Trickey. Plan 9, a distributed system. In *Proceedings of the USENIX Symposium on Microkernels and Other Kernel Architectures*, pages 31–37, April 1992.

[PW93] Dave Presotto and Phil Winterbottom. The organization of networks in Plan 9. In *Proceedings of the 1993 Winter USENIX Technical Conference*, pages 271–280, January 1993.

[Raw97] F.L. Rawson III. Experience with the development of a microkernel-based, multiserver operating system. In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 2–7, 1997.

[RDH+96] J. Rosenberg, A. Dearle, D. Hulse, A. Lindstrom, and S. Norris. Operating system support for persistant and recoverable computations. *Communications of the ACM*, 39(9):62–69, September 1996.

[RMP93] Sanjay Radia, Peter Madany, and Michael L. Powell. Persistence in the Spring system. In *Proceedings of the Third International Workshop on Object Orientation in Operating Systems*, pages 12–23, 1993.

[RNB96] Paul J. Roy, David B. Noveck, and William J. Bryant. State management in a distributed UNIX system. In *Proceedings of the Twenty-Ninth Annual Hawaii International Conference on System Sciences*, pages 170–179, January 1996.

[Rou95] Ellard Thomas Roush. *The Freeze Free Algorithm for Process Migration*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1995.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. On digital signatures and public key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[RTW97] John V. E. Ridgway, Craig Thrall, and Jack C. Wileden. Toward assessing approaches to persistence for Java. In *Proceedings of the Second International Workshop on Persistence and Java*, August 1997.

[RWWB96] Roger Riggs, Jim Waldo, Ann Wollrath, and Krishna Bharat. Pickling state in the Java system. *Computing Systems*, 9(4):291–312, Fall 1996.

[SA95] Jeffrey I. Schiller and Derek Atkins. Scaling the web of trust: Combining Kerberos and PGP to provide large scale authentication. In *Proceedings of the 1995 USENIX Technical Conference*, pages 83–94, January 1995.

[SDH+97] R. Stets, S. Dwarkadas, N. Hardavellas, G. Hunt, L. Kontothanassis, S. Parthasarathy, and M. Scott. Cashmere-2l: software coherent shared memory on a clustered remote-write network. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, pages 170–183, October 1997.

[SESS96] Margo I. Seltzer, Yasuhiro Endo, Christopher Small, and Keith A. Smith. Dealing with disaster: Surviving misbehaved kernel extensions. In *Proceedings of the 1996 Symposium on Operating Systems Design and Implementation*, pages 213–227. USENIX Association, October 1996.

[SG97] D.J. Scales and K. Gharachorloo. Towards transparent and efficient software distributed shared memory. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, pages 157–169, October 1997.

[SI88] Jonathan M. Smith and John Ioannidis. Implementing remote `fork()` with checkpoint/restart. Technical Report CUCS-365-88, Columbia University, 1988. Available at: `ftp://ftp.cs.columbia.edu/pub/reports/reports-1988/cucs-365-88.ps.gz`.

[SN97] M. Satyanarayanan and B. Noble. The role of trace modulation in building mobile computing systems. In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 135–139, 1997.

[SP95] W. Richard Stevens and Jan-Simon Pendry. Portals in 4.4BSD. In *Proceedings of the 1995 USENIX Technical Conference*, pages 1–10, January 1995.

[Spe96] Susan Spence. Distribution strategies for Persistent Java. In *Proceedings of the First International Workshop on Persistence and Java*, September 1996.

[SR98] Volker Strumpen and Balkrishna Ramkumar. *Portable Checkpointing for Heterogeneous Architectures*, chapter 4, pages 73–92. Kluwer Academic Press, 1998.

[SS94a] Mukesh Singhal and Niranjan G. Shivaratri. *Advanced concepts in operating systems*. McGraw-Hill, 1994.

[SS94b] M. Spasojevic and M. Satyanarayanan. A usage profile and evaluation of a wide-area distributed file system. In *Proceedings of the 1994 Winter USENIX Technical Conference*, pages 307–323, January 1994.

[Sun98a] Sun Microsystems. Why Jini now? Sun Microsystems white paper, 1998. Available at: `http://www.java.sun.com/products/jini/whitepapers/whyjininow.pdf`.

[Sun98b] Sun Microsystems. picoJava^{tm}-I microprocessor core architecture. Technical Report WPR-0014-01, Sun Microsystems, From a web page visited October 1998. Available at: `http://www.sun.com/microelectronics/whitepapers/wpr-0014-01/index.html`.

[SY85] R.E. Strom and S. Yemini. Optimistic recovery in distributed systems. *ACM Transactions on Computer Systems*, 3(3):204–226, August 1985.

[Tan87] Andrew S. Tanenbaum. *Operating Systems — Design and Implementation*. Prentice-Hall, 1987.

[Tan95] Andrew S. Tanenbaum. A comparison of three microkernels. *Journal of Supercomputing*, 9(1):7–22, March 1995.

[Tho84] K. Thompson. Reflections on trusting trust (deliberate software bugs). *Communications of the ACM*, 27(8):761–763, August 1984.

[TK95] Anand R. Tripathi and Neeran M. Karnik. Trends in multiprocessor and distributed operating systems designs. *Journal of Supercomputing*, 9:23–49, March 1995.

[TL94] Chandramohan A. Thekkath and Henry M. Levy. Hardware and software support for efficient exception handling. In *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 110–119, San Jose, CA, October 1994. ACM Press.

[TML97] C.A. Thekkath, T. Mann, and E.K. Lee. Frangipani: a scalable distributed file system. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, pages 224–237, October 1997.

[TN98] Anand R. Tripathi and Terence Noonan. Design of a remote procedure call system for object-oriented distributed programming. *Software— Practice and Experience*, 28(1):23–47, January 1998.

[TvRvS+90] Andrew S. Tanenbaum, Robbert van Renesse, Hans van Staveren, Gregory J. Sharp, Sape J. Mullender, Jack Jansen, and Guido van Rossum. Experiences with the Amoeba distributed operating system. *Communications of the ACM*, 33(12):46–63, December 1990.

[UKS95] Ronald C. Unrau, Orran Krieger, and Michael Stumm. Hierarchical clustering: A structure for scalable multiprocessor operating system design. *Journal of Supercomputing*, pages 105–134, March 1995.

[vBM96] Robbert van Renesse, Kenneth P. Birman, and Silvano Maffeis. Horus: A flexible group communication system. *Communications of the ACM*, 39(4):76–83, April 1996.

[vHS97] Peter van Roy, Seif Haridi, and Ralf Scheidhauer. Mobile objects in Distributed Oz. *ACM Transactions on Programming Languages and Systems*, 19(5):804–851, September 1997.

[Wal98] Jim Waldo. Jini architecture overview. Sun Microsystems white paper, 1998. Available at: `http://www.java.sun.com/products/jini/whitepapers/architectureoverview.pdf`.

[WBDF97] D.S. Wallach, D. Balfanz, D. Dean, and E.W. Felten. Extensible security architectures for Java. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, pages 116–128, October 1997.

[Wel94] Brent Welch. A comparison of three distributed file system architectures: Vnode, Sprite, and Plan 9. *Computing Systems*, 7(2):175–199, Spring 1994.

[WKMR96] Jack C. Wileden, Alan Kaplan, Geir A. Myrestrand, and John V.E. Ridgway. Our SPIN on persistent Java: The JavaSPIN approach. In *Proceedings of the First International Workshop on Persistence and Java*, September 1996.

[WMK95] B. Whetten, T. Montgomery, and S. Kaplan. A high performance totally ordered multicast protocol. In *Proceedings of the International Workshop on Theory and Practice in Distributed Systems*, pages 33–57, 1995.

[WO88] B.B. Welch and J.K. Ousterhout. Pseudo devices: user-level extensions to the Sprite file system. In *Proceedings of the 1988 Summer USENIX Conference*, pages 37–49, 1988.

[WPE+83] B. Walker, G. Popek, R. English, C. Kline, and G. Thiel. The locus distributed operating system. *ACM Operating Systems Review*, 17(5):49–70, October 1983.

[WPW+97] D. Wong, N. Paciorek, T. Walsh, J. DiCelie, M. Young, and B. Peet. *Concordia:* an infrastructure for collaborating mobile agents. In *Proceedings of the First International Workshop on Mobile Agents*, pages 86–97, 1997.

[WRW96] Ann Wollrath, Roger Riggs, and Jim Waldo. A distributed object model for the Java system. *Computing Systems*, 9(4):265–290, Fall 1996.

[WWK95] William A. Wulf, Chenxi Wang, and Darrell Kienzle. A new model of security for distributed systems. Technical Report CS-95-34, University of Virginia, August 1995.

[WWW95] Ann Wollrath, Geoff Wyant, and Jim Waldo. Simple activation for distributed objects. In *Proceedings of the First USENIX Conference on Object-Oriented Technologies*, pages 1–11, 1995.

[WWWK94] Jim Waldo, Geoff Wyant, Ann Wollrath, and Sam Kendall. A note on distributed computing. Technical Report SMLI TR-94-29, SunLabs, November 1994.

[WWWK95] Jim Waldo, Ann Wollrath, Geoff Wyant, and Samuel C. Kendall. Events in an rpc based distributed system. In *Proceedings of the 1995 USENIX Technical Conference*, pages 131–142, January 1995.

[YD96] Zhonghua Yang and Keith Duddy. CORBA: A platform for distributed object computing. *ACM Operating Systems Review*, 30(2):4–31, April 1996.

[ZD97] B. Zenel and D. Duchamp. General purpose proxies: solved and unsolved problems. In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 87–92, 1997.

[ZPMD97] Deborra J. Zukowski, Apratim Purakayastha, Ajay Mohindra, and Murthy Devarakonda. Metis: A thin-client application framework. In *Proceedings of the Third USENIX Conference on Object-Oriented Technologies*, pages 103–114. USENIX Association, 1997.

[ZRB+93] Roman Zajcew, Paul Roy, David Black, Chris Peak, Paulo Guedes, Bradford Kemp, John LoVerso, Michael Leibensperger, Michael Barnett, FaraMarz Rabii, and Durriya Netterwala. An OSF/1 UNIX for massively parallel multicomputers. In *Proceedings of the 1993 Winter USENIX Technical Conference*, pages 449–468, January 1993.

[Zwe97] M. Zweiacker. Making CORBA applications fault tolerant using checkpointing and recovery. *Ada User Journal*, 18(2):78–91, June 1997.