G E N I

Global Environment for Network Innovations

Report of the NSF Workshop on Research Challenges in Distributed Computer Systems

Editors:

M. Frans Kaashoek
Barbara Liskov
David Andersen
Mike Dahlin
Carla Ellis
Steve Gribble
Anthony Joseph
Hank Levy
Andrew Myers
Jeff Mogul
Ion Stoica
Amin Vahdat

GDD-05-06

December 2005

Appears as NSF Workshop Report (December 2005)

Status: Final

Report of the NSF Workshop on Research Challenges in Distributed Computer Systems

Editors: M. Frans Kaashoek Barbara Liskov David Andersen Mike Dahlin
Carla Ellis Steve Gribble Anthony Joseph Hank Levy Andrew Myers
Jeff Mogul Ion Stoica Amin Vahdat*

December 4, 2005

1 Executive Summary

This report[1] summarizes recommendations from a workshop on research challenges in distributed computer systems, sponsored by the National Science Foundation. A program committee solicited input from the research community by asking researchers to submit position papers that identified grand challenges in distributed systems, invited researchers based on their submissions, and selected a few position papers for presentation at the workshop. Most of the workshop was organized around break-out sessions, in which we refined the research challenges and identified the facilities needed to carry out future research as well as what the distributed systems community can contribute to the facility. Information about the workshop, including the full program, the selected submissions, and the slides of the presentations and the break-out sessions, is available at `http://www.pdos.lcs.mit.edu/~kaashoek/nsf/`.

The workshop attendees identified a number of challenge applications whose implementation will require research advances in the design and engineering of distributed systems. Examples include: managing a large number of personal devices and data, improving the auto commute through data dissemination and using sensors and actuators in the car to avoid accidents, rapidly deploying fault-tolerant distributed systems to assist in disaster recovery, and understanding and affecting the planet in real-time. Each of these applications has security, storage, fault-tolerance, and usability requirements that can be addressed only if there are new research advances.

To spur these advances, the community has a need for a facility to support experimentation. Facilities such as Planetlab [2] and Emulab [3] have demonstrated that the right facility can spark research progress. The new applications, however, require a scale of facility that is unavailable at present, and that includes product versions of recent research advances. Such a facility would allow researchers to leverage the recent results in tackling the next set of challenges.

The report makes the following recommendations for the research community and NSF. For the research community:

- Use the challenge applications to frame important and challenging research questions in distributed systems. The answers are likely to generate knowledge that goes well beyond the current understanding of distributed systems.

- Participate in the development of a shared facility to experiment with solutions. This development can leverage the recent advances in overlay networks, virtualization, secure global access, resource allocation, and debugging.

- Collaborate with the other communities, such as the networking, sensornet, and security communities, on both collaborative research and on a network infrastructure that is suitable for designing and engineering the challenge applications, exploiting the unique opportunity to rethink the traditional layering from top to bottom.

*This committee was greatly assisted by the contributions of all workshop participants, listed in Appendix A.

[1]The references in this report don't follow the scientific standards for research publications. A few references have been included to allow the reader to easily follow up on some specific points, but the references are not comprehensive.

For National Science Foundation the report makes the following recommendations:

- Sponsor the research identified in this report in upcoming solicitations. In particular, research in security, storage systems, simplifying management, and reliability is likely to lead to the creation of important new knowledge about the design and implementation of distributed systems needed to build the challenge applications.

- Sponsor a shared facility to carry out the research. This facility must include significant storage, has points of presence around the world, has a diverse set of links, and good connectivity to the existing Internet. The facility must also incorporate simulation, emulation, and traces of large-scale distributed systems, allowing researchers to explore isolated questions in controlled ways.

2 New Application Domains

This section identifies distributed applications that go well beyond what is currently possible with the existing knowledge base in distributed systems. The goal of identifying these applications is to frame research problems that can result in novel distributed systems that can support the specific applications, as well as, and perhaps more importantly, unanticipated applications that pose similar challenges.

These applications identified in this report have the potential to improve society and can capture the imagination of researchers. They cover the range of important emerging distributed application domains including networks of sensors and actuators, networked embedded devices, pervasive and ubiquitous computing, disruption-tolerant networks, mobile systems with location-aware applications, real-time acquisition of data, and management of vast amounts of information. Additionally, the example applications have requirements beyond the usual performance requirements, including security, privacy, usability, permanence, availability, flexibility, adaptability, programmability of large numbers of devices, self-monitoring, and self-management.

We identified the following applications:

- **Digital life**. Daily life in the near future will surround us with digital devices mediating many activities at work and at home, often invisibly affecting our environment and connecting with each other and with the global Internet. While the digitally enhanced environment can offer significant benefits, the complexity of managing large numbers of interacting embedded devices and the potential threats to security and privacy must not become burdens and risks to the user. Special instances of enhanced digital living applications include health monitoring and the management of our personal data collections:

  – Health monitoring. Technology can give people more independence and mobility within the community even when they have a serious health problem. The ability to monitor the health of at-risk patients beyond just the home environment and to notify doctors or family of medically-relevant events (ranging from ones requiring immediate intervention to ones contributing to diagnosis) while protecting privacy and security of such information can allow a person to more safely pursue a normal lifestyle. Such an application involves sensors, possibly actuators, location-awareness, disruption-tolerant wireless communications interfacing to the Internet, and secure medical data.

  – Managing vast personal data. Individuals can generate large amounts of data, including digital photographs, music, and documents. They acquire even more by sharing such data with others. How does anyone locate that special photo of a family reunion several years ago among thousands stored with camera-generated filenames? Users need tools to manage their individual data, to control how sharing is done, to ensure preservation of precious family records over generations, and to organize and retrieve desired data easily. Data about individuals are also captured by others during business transactions as well as casual activities. How can people understand and have some control over what data are being disseminated and stored about them?

- **Managing the auto commute**. The daily commute to and from work is one of society's great aggravations and inefficiencies. Each driver makes local decisions based on the information available at that car. Networked data dissemination can significantly improve the timeliness and scope of that information. Sensor and actuators in the automobile can unburden the driver and help avoid accidents. Overall traffic flow could be more effectively modeled, analyzed in real-time, and managed. The vision is of a safer, more energy efficient, and less annoying commute in spite of increasing traffic loads.

- **Disaster recovery.** Communication breakdowns and loss of vital information in a disaster scenario are critical problems that can prevent an effective response. During a disaster, it is essential to rapidly deploy a replacement communications network and keep it running as long as necessary. The affected area may lack basic infrastructure and services such as power, transportation, and security, and the network must be robust to those conditions in order to support life-critical communication services. Lost data storage must be restorable from geographically distributed replicas. Reducing the complexity of setting up the portable network infrastructure and providing the redundancy that allows data recovery depends on effective self-management of these systems.

- **Understanding and affecting the planet in real-time.** The ultimate challenge is to understand the interdependent dynamic systems of our planet—its climate, politics, populations, ecology, economy, etc. Instrumentation can provide real-time data and large-scale computation can combine these distributed data streams to extract global meaning to, for example, assist in energy management. This visionary global-scale application can drive innovation in resource management, security of information, and network design.

What is missing in our knowledge of distributed systems that is needed in order to deliver the functions of these applications? We identified the following high-level challenges:

- 10^6 plug-and-play devices. If we are to be surrounded by interconnected embedded devices in our digital lives, users can't be expected to explicitly configure and maintain all of these devices. We will need the ability to automate and provide self-management of such devices.

- Balancing security, usability, and flexibility. It is clear there are tradeoffs among these goals, but the nature and impact of the tradeoffs are not understood well by system builders or by users.

- Policy expression. Our capability to articulate a policy in areas like security or privacy is limited.

- Integrating evolution into our distributed systems. Our systems need to be resilient and adaptable to change in order to provide the unprecedented longevity of continuous service demanded by future applications. Change can take many forms: software upgrades to long-lived services, obsolescence of data formats, changing user behavior patterns and expectations, the need to retarget real-time data acquisition in an already deployed sensor network, and other unanticipated changes. It may be necessary that systems must run with a mixed of old and new hardware and software, because older parts may be difficult to upgrade.

- Five nines availability. Since some of our example applications deal with life-and-death situations, the infrastructure and distributed systems addressing them must provide levels of availability and disruption-tolerance that they currently cannot achieve.

- Vast amounts of personal storage. The increasing importance of digital personal storage raises research issues of tracking data capture, providing permanence of data storage, and enabling easier data retrieval with a heterogeneous and evolving collection of devices, systems, services, formats, and networks.

- Projection of the physical world into the digital world. Linking the digital and physical worlds as required by several of our target applications requires techniques for representing and naming physical objects.

- Finding information—to search everything. Searching data that scale in volume and evolve with time with a usable interface is a difficult problem. When one adds the mapping of objects in the physical world into the digital world, one can envision generalizing the search problem to finding anything - including one's car keys. Finally, how does one ”find” a specific person to be contacted instead of explicitly choosing among the multitude of communication devices and networks that might be associated with that user?

3 Research problems

This section categorizes the challenges identified in Section 2 by research area and expands on them by identifying specific research challenges.

3.1 Security Challenges

Everyone knows that current distributed systems offer too little security and endanger privacy. Yet these systems are increasingly important to almost every activity, and a serious, widespread failure of security could be catastrophic. Furthermore, as reliance on the Internet grows, and networked computing becomes more and more pervasive in our lives, the situation will only get worse.

New research is needed to make the global distributed computing infrastructure secure. The techniques that result from this work can enable important applications that are otherwise infeasible. For example, integrated medical information systems would allow better medical care, but they must protect patient privacy. The traffic control system mentioned in Section 2 similarly requires protection of privacy and also anonymity, since otherwise a citizen's movements could be tracked.

New research is needed on all important security problems, e.g., to prevent phishing attacks, identity theft, worms, and denial of service attacks. Here are some that in particular require attention:

- Making networks secure. The design of the Internet creates intrinsic vulnerabilities to malicious hosts. For example, attacks such as flash worms and distributed denial of service are made easier because any host on the Internet can construct an IP address and command the routing infrastructure to direct packets to that location. Other key services such as the Domain Name Service are similarly open to abuse. If hosts may be malicious, the network must enforce some security requirements from below. The basic assumptions of the current network architecture need to be reconsidered; for example, the use of routing and name resolution can perhaps be mediated by access controls while preserving functionality.

- Confidence in the environment. In an increasingly mobile, changing world in which computation is done using small, pervasive computing devices in wireless networked environments, there is an intrinsic question of whether the device or the environment in which the computation runs can be trusted. This issue arises, for example, whenever a user uses a shared (e.g., public) computer to access sensitive data, because the shared computer may be compromised to leak data it accesses. New methods for obtaining trust in a computing device and its environment are needed. For example, a trusted platform module based on attestation may offer a better and more flexible way to obtain trust in the environment.

- Usability. Many security vulnerabilities arise because users do not understand the implications of their decisions, whether these involve choices in configuring their local machine, or simply the choice to click on a URL. New approaches are needed that simplify and automate security management, particularly in large, complex distributed systems. One promising research area is new approaches for exploring and visualizing security policies: “wizards” that can help improve the decisions that users must make about what policies to use and what to trust. Another approach is a drastic reduction in the amount of configuration users must perform, since many attacks exploit configuration mistakes.

- Incentive-based security. Complete prevention of bad behavior may be infeasible. A promising alternative approach is to use incentives that reward good behavior and punish bad behavior. If participants run a significant risk of punishment for improper actions, rational participants will have an incentive to behave better. In general, if participants have some investment in using a system, it is possible to give them incentives to behave by the rules. For example, eBay manages to deter bad behavior because participants are invested in their reputation; bad behavior hurts their reputation and makes further transactions difficult.

- Auditing. To recognize security breaches, to be able to punish intruders in court, and also to support incentive-based security, we need to track what users do and store this information in an audit trail. The tracking mechanism should be implementation-independent, so that different applications can share it; thus we avoid duplication of work. One major issue with such a mechanism is scale: there is a great deal of behavior to keep track of, and fine-grained information is needed to capture what is really going on. To improve scalability, new techniques for abstracting and compressing system state and history are needed, because it would be prohibitively expensive to keep a complete record of all system behavior (e.g., including all network packets). A second major issues is how to monitor behavior while protecting privacy. Clearly, audit logs could be mined to violate user privacy, yet accuracy is required for the logs to be useful. For example, suppose a user is negligent in defending his computing resources. To recognize this requires recording enough information about the evolution of the state of a computer so that a vulnerable, poorly maintained computer can be identified, yet this must be done without violating the privacy of every computer user. Thus new methods that balance privacy and and accuracy are needed.

- Software support. Systems today are implemented at a relatively low level of abstraction that supports the classic host-based view of a distributed system, in which each host is in charge of just its own security. As the computing infrastructure evolves toward pervasive, embedded communicating devices sharing information, this programming model is increasingly out of sync with the way that distributed systems are constructed. Higher-level RPC-based interfaces, such as Java's Remote Method Invocation, hide some of the low-level mechanics, but tend to hurt rather than help with assessment of system trustworthiness: they remove needed control from the programmer and obscure security-critical details. A higher-level programming model is needed that will make construction of trustworthy distributed systems easier in a "post-host" world.

- Security validation. Building trustworthy distributed systems is challenging. These systems must be robust in the face of misuse or attacks, remaining available while preventing damage to critical information and leakage of confidential information. Individual techniques exist that help to enforce these integrity, availability, confidentiality, and consistency properties. But it remains difficult to build a high-assurance distributed system using these techniques in combination. Work on validation of security and fault tolerance has typically examined particular kinds of computing systems, with respect to a limited set of system-level properties. More complete techniques are needed for validating all aspects of system security in distributed systems.

3.2 Storage

Data storage is a fundamental challenge for large-scale distributed systems, and advances in storage research promise to enable a range of new high-impact applications and capabilities. Illustrative examples include:

- Data sharing for agile organizations. Hundreds or thousands of workers from several relief organizations securely share highly-available emergency response data among themselves and with victims after a regional disaster. Employees from several organizations collaborate on a joint project using a secure, shared data workspace.

- Self-managing storage. A large-scale storage system reliably and securely stores highly-available data for an enterprise but the only maintenance it requires is that new racks of disks are added and old ones removed as storage technology advances over the course of years.

- Personal storage. An extended family shares selected photos and videos with each other and with friends securely and over decades, with no information lost to disk failures, worm/virus attack, or careless operation of the system.
- Real-time data streams. Researchers search for, access, and analyze real-time data streams from network sensors.
- Perfect memory. Non-expert users are able to recall any word they have read, written, heard, or spoken at a relevant moment when that information is useful.
- Access-anywhere storage. A user with dozens of data-access devices (phone, camera, PDA, laptop, car, music player, ...) is able to access any of her data from any device at any time and from any location.
- Medical data. Medical researchers analyze anonymized data for millions of patients across hundreds of electronic medical record storage systems in order to retrospectively analyze the effects of different treatments or to prospectively identify outbreaks of disease.

Although these examples only scratch the surface of the opportunities enabled by ubiquitous access to ever-increasing amounts of increasingly valuable data, nevertheless a set of cross-cutting research issues arises from them:

- Long-term durability. We lack understanding of how to architect storage systems that reliably store data for decades and we lack the ability to model or predict the reliability of a storage service or architecture over such timescales in the face of media failures, operator errors, malicious attack, business failures of storage service providers, and format evolution.
- Retrieval and interfaces. The venerable “files and folders” interface for organizing storage appears not to be the right model for a world with many more sources of non-traditional data, enormous amounts of data, non-expert users, and non-traditional applications.
- Auditability and provenance. New applications and regulatory requirements make it increasingly important to be able to track the original sources of data, who modified data, who read data, and what actions depend on what data.
- Managability. Increasing volumes of data, increasing numbers of devices, and visibility of digital data to non-expert users make it essential to vastly reduce management cost. In particular, techniques requiring effort that is linear (or worse) in the size of storage or the number of storage devices will not scale.
- Security and sharing. Increasing diversity of applications and users make it increasingly important and difficult to share data conveniently and securely.
- Fundamental trade-offs. Fundamental trade-offs of consistency v. availability v. performance demand new design trade-offs for new applications, new hardware capabilities, and new workloads.
- Meeting high-level goals. Given any set of high level goals for a system’s durability, consistency, performance, resource constraints, and availability, a system should automatically adjust its replication strategy to meet its goals.

3.3 Simplifying Management

The advance of cheap, high-performance processors has made general-purpose CPUs ubiquitous in our everyday lives: from control and audio-visual devices in our homes, to the communications devices we carry in our pockets, to the tools we use at work, to the large-scale clusters that drive the global-scale services we rely on. Unfortunately, the computer-based automation of our lives has created a major source of stress for the majority of the population. While technology has the potential to simplify our lives, it also frustrates us because of the complexity of the devices we use and the common problems we face in using them. Overall, computing and embedded devices are often prone to failure, and unintuitive and difficult to use and manage. As the number of devices grows, the difficulty of coordinating their interactions grows as well.

Here we briefly examine two environments, the home and the enterprise, and look at a set of research challenges whose solution could greatly simplify management in both environments. To begin with, homes are becoming increasingly complex, to the point where managing technology is a major challenge for most homeowners. Most homes already have a large (and expanding) set of heterogeneous hardware devices, such as desktop and laptop computers, audio/visual and entertainment devices, and control devices (e.g., lighting, heating, security, etc.). These devices include a heterogeneous collection of software systems, including commodity operating systems, "hidden" operating systems (e.g., inside an Xbox or TIVO), and embedded control software (e.g., inside a thermostat). Furthermore, there will be a growing set of services provided to the home over the Internet, including storage and backup, monitoring, and entertainment.

For the homeowner, this environment with its large set of software and hardware components can be a nightmare. For example, homes must be protected from malicious threats on the Internet. How does a naïve user know how to set firewall policy for their firewall or NAT box? Or, how do parents control content for their children, across the complete set of computers and other devices? Overall, users will want to know exactly what content is entering and leaving the house, and this is difficult, given the variety of communications connecting a modern home (e.g., phone, wifi, cable, dsl, cellular, etc.). Finally, once everything is configured and working, how does a homeowner reconstruct the environment should he or she move to a new house?

The enterprise environment can be exceedingly complex, even for the professional IT managers who are tasked with controlling it. A modern enterprise includes large numbers of computers, software systems, devices, and possibly sensors. There are typically different versions or generations of hardware systems, software systems, and network infrastructure (routers, firewalls, etc.). To an increasing extent, a modern enterprise uses off-the-shelf applications for running their business; different applications are purchased from different vendors, yet these applications need to interoperate successfully, both within the machine room and across the Internet. As well, a company may have sharing relationships with other enterprises, allowing those enterprises access to some but not all of its data. Finally, employees will typically need access to internal data and processes through mobile devices deployed outside of the company's firewall.

For the IT manager, the enterprise computing environment is a constant challenge. Setting policies and ensuring that they meet business rules is hard, and it is often difficult to maintain consistent policies across a widely distributed organization. Software is difficult to manage: installation, configuration, update, and retirement of software executing on a large body of machines is tedious and error-prone. Access control critical for ensuring privacy and security is often difficult to specify and implement in a heterogeneous environment. While there are many software packages that focus on problem reporting, diagnosing problems (as opposed to just noticing that something is wrong) is left to human experts. Testing within the enterprise is difficult, due to the scale of deployment; it may be necessary to reserve hundreds or even thousands of machines just for test purposes in some environments.

The research goal of this effort is to greatly simplify management of technology across the various domains. For example, a user at home should be able to purchase a new set of devices, plug them in, and have them automatically configure themselves, not just individually, but for operation with all the other devices already in the home. In the enterprise, an IT manager should be able to easily create policies and verify that those policies meet requirements. In both domains, software should be easy to install, control, update, and remove.

There are a number of research tasks whose completion will result in simplifying the management of these environments, including the following:

1. Creation of tools for easily specifying and verifying policies for security, privacy, and information control.

2. Creation of automated tools that drive configurations from those policies.

3. Creation of better tools for object access control in complex heterogeneous environments.

4. Better automated component (software) installation and life cycle management, to simplify the processes of updating software and removing outdated software.

5. Creation of mechanisms for global visibility, control, and coordination of components (both hardware and software) within an environment.

6. Better protocols for devices to learn about and communicate with each other, so that they can interact dynamically in the environment.

7. Facilities to create virtual communities, allowing individuals or organizations to communicate and collaborate flexibly.

8. Creation of tools that allow users to assess what the result of some action would be if they took that action; e.g., "Tell me if something I'm going to do will work," or "Tell me the infrastructure cost of a policy change I'm about to make."

9. Automated instruction, monitoring, and fault and event analysis systems, including correlation of events.

10. Time travel facilities that cross multiple hardware/software platforms, allowing backup to previous states.

3.4 24x7

Network services have become critical to national and international infrastructure. However, these services haven't achieved the same number of 9's as other infrastructure services. For example, the Internet achieves between 2 and 3 nines of availability, while the telephone network achieves between 4 and 5. Worse, as the complexity and importance of network services grows, it will be a challenge to maintain the current level of availability.

To provide continuous operation in such a complex environment requires an *holistic* approach. Delivering continuous operation is not about any individual computer, component, or system operator, but rather about the service as a whole. Failures are inevitable; ensuring continuous service availability in the face of failure is the challenge.

To address this challenge research is needed in the following areas:

Failure Models: Most work on fault-tolerance has assumed that failures are either failstop or Byzantine. An important question is whether there are additional failure models in which the failed nodes exhibit behavior in between these two extremes. For example, some recent work has considered failures in which nodes are merely "selfish": they misbehave only when there is some rational advantage to be gained. Intermediate failure modes are interesting if they (1) correspond to reality and (2) allow the use of more efficient fault-tolerance techniques (e.g., replication algorithms that require fewer replicas or fewer rounds in their protocols).

Avoiding Correlated Failures: Work on fault-tolerance also assumes that failures are uncorrelated, but this may not be true in reality. A particular problem is failures due to deterministic errors in software, which can cause all replicas in a group of machines to fail simultaneously. N-version programming, in which each replica runs a different version of the software, is generally an impractical approach because the versions must be truly independent; this condition is met only for a few services (such as file systems), and even in these cases, relatively few versions exist.

An alternative is to try to avoid deterministic errors by manipulating the code image so that, for example, a buffer overrun won't cause all versions of the code to fail. The approaches that have been investigated so far, however, have limited applicability; developing new approaches with wider applicability would be very useful.

Living with Failure: In spite of our best efforts we must assume that failures will happen. One approach to coping with this problem is to attempt to mask failures by recognizing them quickly and then causing the failed node to recover very fast. Some success along these lines has been achieved in the work on micro-reboots; more work in this area would be very useful.

Failure Detection: In general we would like systems to monitor themselves so that failed nodes can be automatically removed from service. It is easy for automatic monitoring to detect nodes that have failed in a failstop manner: just probe them long enough to rule out the possibility that the lack of response is due to communication problems. However it is unclear how to detect Byzantine-faulty nodes, since they are capable of responding to probes in the appropriate manner even though they are failed. Techniques that allow such failures to be detected are needed.

A related problem is determining whether a response from a single server is based on your request being processed by the code you intended to use. Some progress along these lines has been made by the work on software attestation, but better techniques are needed.

Self Configuration: We would like systems that do not rely on human operators to reconfigure, both because this is a well-known source of errors, and because with a human in the loop, reconfiguration will be slow. Furthermore, given failure detection, a system can know when nodes need to be removed from service, and their tasks distributed to other nodes. However, there are many issues to work out to achieve a practical self-configuring system.

Software Upgrades: In a long-lived system, we must expect that the software will need to be upgraded to improve performance or provide new features. Techniques for allowing software upgrades to be installed automatically are needed. One issue is that the upgrade may need to take effect gradually, and there may be long periods of time when different nodes in the system are running different, possibly incompatible, software versions; techniques are needed to allow the system to continue to provide service under these conditions. Another issue is that the upgrade may contain incorrect code, and as a result it may be necessary to roll it back to the previous version. Techniques to allow such roll backs are also needed; this is a hard problem because the system must not lose data that came into existence prior to the roll back.

Better Tools: In spite of programmers' best efforts, errors exist in their software systems. Tools that can help avoid such errors can greatly improve the reliability of systems. The tools might take the form of a platform on which to build distributed services; the platform would provide a rich functionality, thus reducing the work for the system developers, but must avoid hiding features that developers need. Or the tools might encorporate new ways of finding errors, either during the software development process, or after the fact, while the system (possibly consisting of legacy code) is running.

Availability versus Consistency: Delivering continuous operation in the face of failure typically requires replication of system state. Replication introduces known tradeoffs between data consistency and availability. On the one hand, a system could strive for consistency, waiting to update a sufficient number of data replicas for writes or to contact a sufficient number of replicas for reads before returning successfully. On the other hand, the system could return without waiting to communicate with a sufficient number of replicas. Experience with replicated systems suggests that perfect consistency results in unacceptable availability while striving for the highest level of availability results in unacceptable consistency. While there have been various efforts to quantify and control the tradeoffs in this space, none has exported a practical programming model to system developers.

4 Infrastructure for research

To evaluate what facilities are needed for carrying out the identified research, we considered the challenge applications discussed in Section 2, as well as the research challenges discussed in Section 3.

We have identified two distinct kinds of facilities needed to support our research. First is a distributed facility that can be used to evaluate new distributed systems in a realistic environment. Second is a local facility that allows experimentation with new systems under carefully controlled conditions. The first kind of facility is exemplified by Planetlab today, while the second is exemplified by Emulab. An important difference between the two environments is that many systems can be running on Planetlab nodes simultaneously, whereas in Emulab a researcher has complete use of a subset of machines to run his or her experiment.

In either case, the facilities require both physical hardware resources and the software artifacts to allow users of the facility to manage and access it effectively.

The rest of this section discusses the required facilities in more detail. We answer three questions: (1) what does our community need from a facility? (2) how can our community contribute to realizing such a facility?, and (3) what services can our community provide on this facility (that will be useful not only to us but to other communities as well)?

4.1 What does the community need?

To address the challenge applications, the community needs a facility that includes significant storage, has points of presence around the world, has a diverse set of links, and good connectivity to the existing Internet. The facility must also incorporate simulation, emulation, and traces of large-scale distributed systems.

A thread through all the challenge applications is how to handle large amounts of data, and thus the facility must support experimentation with large amount of data. To experiment with real-world failures and to be able to handle them, the facility should have multiple points of presence across the world and a diverse set of network links. To be able to attract Internet users so that the community can observe real workloads, the facility must have good access to the Internet. A good target for the shared facility is:

- 20-30 sites across world
- a cluster of at least 256 machines per site
- at least 256 TB per site

In addition, the community could use a sizeable number of small clusters with heterogeneous connectivity, and the facility must have sufficient resources to support the services outlined in Section 4.3; in particular, logging networking events may require a large amount of storage and tools for processing the logged information may require a large amount of computation.

Many applications require, in addition to "common" resources, a set of application-specific resources and the artifacts to make those resources useful. Identifying and creating a subset of these application-specific facilities will be necessary to allow users of the facility to create and deploy many of the challenge applications. These application-specific facilities include:

- Vehicular networks
- Environment / habitat monitoring
- Home networks and consumer devices
- Human telemetry data monitoring
- "Digital life" facilities (personal data production)

A important requirement for the facility is evolvability. Facility users should be able to easily add and remove their resources to and from the facility. For instance, a user of the facility should be able to add her own cluster to the facility for the duration of her experiments or share her cluster with a subset of other facility users. Furthermore, users should be able to integrate into the facility entire networks such as sensor networks, wireless networks, and community networks. Evolvability would allow the facility to grow organically, as the facility users need changes or new applications emerge.

In addition to a real-world, distributed facility, the community also needs support for simulation and emulation. Emulation needs to be done in an environment in which the experimenter can control all aspects of the experiment; this requires a separate facility from the real-world facility, since sharing of resources doesn't allow the kind of control that is needed. Simulation is necessary to experiment with "larger" things than what can be done for real, e.g., huge numbers of users, very large periods of simulated time.

The recently proposed GENI facility [1] is a good starting point for fulfilling the support requirements for the challenge applications and the research issues identified in this report. Because of its cross-disciplinary nature, the GENI facility provides a unique opportunity for the distributed systems community to influence future network designs and to enhance the facility by contributing recent advances in distributed systems.

4.2 How can the community contribute?

To be really useful, the facility needs to be accompanied by software. In this section we discuss the software that is needed to make the new facilities usable. The distributed systems community has expertise in providing this kind of software.

Virtual system network. The design of any shared facility has to balance between (1) efficient resource sharing, and (2) providing predictable performance. To achieve efficient resource sharing, one needs to virtualize all facility resources including bandwidth, CPU, memory, and storage. On the other hand, achieving predictable performance would require one to provide the user with the abstraction of a dedicated facility. With such an abstraction, a user of the facility can specify a virtual network topology by associating a desired bandwidth and delay with each virtual link, and specify the computation and storage resources at every node.

To implement the required virtualization, we need to address two challenges. First, given the specification of a system network, we need to allocate the appropriate resources in the facility to implement it. This problem is an instance of the constrained distributed resource allocation problem, which is NP-complete. While several heuristics have been proposed to solve this problem, we may need to develop new ones to take advantage of the particular instance of our problem. Second, we need to enforce resource allocation on each shared resource. One way to achieve this would be to use virtual machines in conjunction with proportional share allocation schedulers for the CPU, and weighted fair queuing algorithms for allocating the bandwidth along virtual links.

Fair resource allocation. Another important problem that needs to be addressed in the context of a shared facility is how to allocate resources among the users of the facility when the facility is oversubscribed. Existing facilities use simple policies that partition the facility either in time or space. For instance, with PlanetLab, each facility user gets a slice on each machine in the facility, and each machine divides its resources proportionally among the competing slices. As a result, the performance seen by a user degrades as the number of active users of the facility increases. In contrast, with Emulab, the resources are statically partitioned in space, with each facility user having full control of a set of machines. This allows Emulab to provide highly predictable performance to its users, but at the cost of long waiting times during the oversubscription periods.

These limitations, which will be only magnified as use of the facilities increases, have prompted researchers to look elsewhere for more flexible resource allocation models. One approach that has gained traction recently is using a market-based approach, where each user of the facility receives some virtual money, which can be used to "buy" computation, communication, and storage resources. Furthermore, if no more resources are available, facility users may be able to trade resources among themselves. The distributed system community has a long tradition in developing market and economic-based resource allocation schemes. This experience puts our community in an excellent position to develop the resource allocation policies and techniques for the shared facility.

Support for auditing and debugging. A major challenge in managing a large scale facility is to identify misconfigurations, and also potential attacks on the facility or attacks initiated by machines in the facility. A related challenge is debugging distributed applications, a notoriously difficult and time consuming task. Recording detailed information about the execution of each application would go a long way to address these challenges. Indeed, imagine that each node logs all incoming and outgoing packets and performs periodic virtual machine checkpoints. Such information would be very useful in debugging applications and in identifying attacks, because of the ability to determine communication patterns and/or packet content.

Gathering such detailed information is a daunting task, as it requires not only a huge amount of storage, but also the ability to efficiently search the data. There are many trade-offs that one could make in designing such a logging system to simplify the problem. For example, one could log only the packet headers, or checkpoint less frequently the virtual machines that are inactive (indeed, there is little need to checkpoint a virtual machine that is idle).

While we believe that a basic logging system that always logs but at a predefined granularity should always be turned on to allow auditing, it is unlikely that the level of detail provided by such logging will satisfy all facility users. For instance, some facility users may want to log the content of the packets (if the basic logging system doesn't), or log other events such as timeouts and interrupts. To support this need, we have to develop a logging framework that is extensible and customizeable. Since developing such framework requires expertise in a variety of fields including operating systems, storage, and networks, the distributed system community is in a unique position to address this challenge.

Simulators. The distributed systems community is good at building high-performance single-machine simulators using virtual machine technology. This line of research needs to be extended to building simulators for complete distributed systems, with realistic fault models. The simulators must allow very large scale simulation, since one reason for simulation is to determine whether a design and implementation meets its goals at scales larger than you can actually test, over longer time frames, with loads, faults, and changes that aren't normally seen.

The community is also good at collecting data and making it available in a way that is suitable for use in simulators. The community can provide software to collect and store topologies, fault loads, and change loads, with increasing realism (more detail) and evolving as the world changes.

4.3 What services can the community provide?

This section discusses services that would be deployed to run on the facility. We have identified two categories of such services that our community can develop and deploy on the facility: *basic* services, and *generic* services. To support these services, additional resources may be necessary, in particular if some of the services become popular over time.

4.3.1 Basic services

Basic services are the services necessary to carry out research experiments on the facility. One example of a basic service is an improved Domain Name Server (DNS) that provides improved resilience and allows fast updates. Another example of a basic service might be a discovery system to locate lightly-loaded and/or nearby resources.

Here we discuss a few other example, in more detail:

A Distributed Authentication Infrastructure. A Public Key Infrastructure (PKI) and a Certificate Authority would allow strong identities for the facility users. Authentication is required for both the network facility itself, to grant access to applications and services and provide a basis for resource isolation, but also for applications and users. A flexible and accessible public-key or other authentication service, along with the software and resources to manage it, will bootstrap both the network itself, and the development of applications on top of it. This service must include the development of libraries to allow a variety of applications to use the service and the development of guidelines for how and when applications should use the service.

Tools for developing distributed systems. Robust and easy to develop applications and services are the fundamental building block of a successful network. Application and and service developers—including researchers—will benefit from the development and availability of tools to help build distributed systems. These tools include language and compiler support, program and protocol analysis tools, and methods for program verification and validation; such tools are discussed in Section 3.4.

Global data access. A single experiment will often involve many nodes and shared data. The basic services should make it easy for programs running on different machines to access and share data in a location-independent manner. Example services in this category include distributed file systems, distributed hash tables, and facilities for flexible logging and data collection.

4.3.2 Generic services

Researchers often deploy their relatively mature systems on the shared facility for extended periods of time in order to gain understanding of their behavior under realistic loads.

It may be appropriate to allow certain of these systems to run on the facility indefinitely. This is appropriate for services that are useful to large user communities, including non-computer scientists. Examples of systems that might be allowed to run indefinitely are: Citeseer, SourceForge, a conference submission service, Fastlane, spam filters, distributed firewalls, and an open search engine.

We must be careful in deciding to allow mature systems to run on the facility indefinitely, since the more of these systems are supported on the facility, the less available the facility becomes for its original use of experimenting with new research systems.

5 Summary

This report has 4 major conclusions:

- Research in the challenge applications, or in other applications of a similar nature, is likely to generate new knowledge in the engineering of distributed systems.
- Effective implementation for these distributed applications requires research in many areas, including most aspects of security; the engineering of massive, searchable storage systems; the engineering of systems that operate 24x7 in the face of a wide range of failures; and the engineering of systems that are easy to use and manage.
- A shared facility would be a enabler of this kind of research.
- Recent research results in distributed systems in the areas of virtualization, resource allocation, auditing, simulators, and transparent and secure data access should be included in the shared facility since they can significantly enhance its usability.

References

[1] National Science Foundation. The GENI initiative. `http:/http://www.nsf.gov/cise/geni/`.

[2] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the I nternet. In *Proc. of the 1st HotNets Workshop*, Oct. 2002. `http://www.planet-lab.org`.

[3] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, Dec. 2002. USENIX Association. `http://www.emulab.org`.

Appendix A: Participants

Andersen, David	CMU	dga+@cs.cmu.edu
Anderson, Tom	U. Washington	tom@cs.washington.edu
Arpaci-Dussean, Remzi	Wisconsin	remzi@cs.wisc.edu
Castro, Miguel	Microsoft	mcastro@microsoft.com
Chase, Jeff	Duke	chase@cs.duke.edu
Comer, Doug	Purdue	comer@cs.purdue.edu
Dahlin, Mike	U. Texas	dahlin@cs.utexas.edu
Druschel, Peter	Max Planck	druschel@cs.rice.edu
Ellis, Carla	Duke	carla@cs.duke.edu
Fleisch, Brett	NSF	bfleisch@nsf.gov
Gribble, Steve	U. Washington	gribble@cs.washington.edu
Jannotti, John	Brown	jj@cs.brown.edu
Joseph, Anthony	Berkeley	adj@cs.berkeley.edu
Kotz, David	Dartmouth	dfk@cs.dartmouth.edu
Lepreau, Jay	Utah	lepreau@cs.utah.edu
Levy, Hank	U. Washington	levy@cs.washington.edu
Mogul, Jeff	HP	Jeff.Mogul@hp.com
Myers, Andrew	Cornell	andru@cs.cornell.edu
Pai, Vivek	Princeton	vivek@CS.Princeton.EDU
Parulkar, Guru	NSF	gparulka@nsf.gov
Peterson, Larry	Princeton	peterson@cs.princeton.edu
Reiter, Mike	CMU	reiter@cs.cmu.edu
Roscoe, Mothy	Intel	timothy.roscoe@intel.com
Roussopolous, Mema	Harvard	mema@eecs.harvard.edu
Satya	CMU	satya@cs.cmu.edu
Schwan, Karsten	Georgia Tech	schwan@cc.gatech.edu
Shah, Mehul	HP	mehul.shah@hp.com
Shenker, Scott	Berkeley	shenker@cs.berkeley.edu
Shrira, Liuba	Brandeis	liuba@vilnius.lcs.mit.edu
Stoica, Ion	Berkeley	istoica@cs.berkeley.edu
Terry, Doug	Microsoft	terry@microsoft.com
Touch, Joe	ISI	touch@ISI.EDU
Vahdat, Amin	UCSD	vahdat@cs.ucsd.edu
Verissimo, Paulo	Universidade de Lisboa	pjv@di.fc.ul.pt
Vogels, Werner	Amazon	werner@amazon.com
Wallach, Debby	Google	kerr@google.com
Weihl, Bill		bill@weihl.com
Welsh, Matt	Harvard	mdw@eecs.harvard.edu