Identifying Open Problems in Distributed Systems

Andrew Warfield, Yvonne Coady, and Norm Hutchinson
University of British Columbia
{andy, ycoady, norm}@cs.ubc.ca

The technology above and within the Internet continues to advance, and has reached a point where the potential benefits of very large scale, finely distributed applications are more apparent than ever. Opportunities are emerging to develop large systems that cater to highly dynamic and mobile sets of participants, who desire to interact with each other and stores of online content in a robust manner. These opportunities will inevitably dictate a substantial body of research in the years to follow. Although applications intended to function at this scale have recently begun to appear, there remain a broad set of open problems that must be faced before this emerging class of distributed system can become a reality.

1 Introduction

Distributed systems research has historically avoided many hard problems through the carefully calculated use of operating constraints. Scalable resource clusters are assumed to be tucked away in protected facilities and connected by reliable infrastructure [15]. Large systems are assumed to have cooperating nuclei of administrative organizations that do not fail [8]. In peer environments, participants are assumed to behave fairly instead of leaching resources [3]. As the specifications of these systems grow to require operation at a massive scale with highly distributed administration, these assumptions will be strongly challenged as a means of providing useful systems. In short, distributed systems research is quickly approaching a point at which many hard problems cannot be avoided any longer.

Prior to embarking on the construction of a large-scale distributed operating system, we felt that it would be useful to survey the landscape of problems that will be faced in the construction of this class of system. This paper is a summary of urgent problems that must be addressed in order for successful systems of this caliber to be realized.

Our approach to identifying open problems is twofold. First, we have designed a taxonomy to describe the domain of existing and future distributed systems. This model is a two-dimensional space whose axes define (1) the concurrency and conflict of resource access, and (2) the degree of distribution and mobility of resources within the system. From this model, we draw four phyla of application: point-to-point, multiplexed, fragmented, and peer-to-peer. This last phylum defines our target domain and we apply lessons learned from the other three groups to it. Through our taxonomy, we describe a set of architectural systems problems that must be addressed.

The second aspect of our examination has been to step back and examine the implications involved with the adoption of large-scale distributed operating environments. In this section, we are less concerned with classical systems issues (performance, robustness, and scale) and more concerned with pragmatic factors involved in building a good system. We present a broad set of pertinent problems that will need to be addressed for these systems to be successful outside of the research laboratory.

2 Taxonomy of Distribution

This section presents a taxonomy, describing four phyla of distributed systems in a continuous space along two axes. The axes, access concurrency and resource distribution, stem from an examination of the evolution of distributed applications. Access concurrency considers the number of simultaneous accesses to a resource and the degree of conflict between these accesses. Access concurrency problems emerged as researchers began to move towards time sharing on mainframes. Resource distribution represents how broadly a system is spread across a network infrastructure.

Individually, each of these axes represents a steadily increasing gradient of complexity within system architecture. It is in the cases where both axes have high degree that system complexity explodes. Indeed, distributed applications seem to all reside very close to the axes in our models. This observation suggests that there must be some limiting factors that exist, inhibiting the development of complex systems.

We now consider the two axes and four phyla of systems individually.

2.1 Access Concurrency

Access concurrency originated with the desire to allow users to share the resources of original mainframe computers. Concurrency mechanisms allow clients to share a resource while preserving the state of that resource during simultaneous accesses. It is worth nothing that without a requirement to avoid conflict, concurrency mechanisms need only act as stateless request multiplexers. Although there are complexity issues in simple multiplexing at the Internet scale, it is conflict avoidance that makes access concurrency especially hard. In order to avoid conflicts between concurrent access, extra mechanisms must be put in place. These mechanisms add overhead and complexity to the system.

Mechanisms to support access concurrency involve tradeoffs between efficiency and effectiveness. Very efficient concurrency control techniques aim to allow the highest possible amount of simultaneous access, but may do so at the cost of poorly preserving resource state or unfairly scheduling this access. Techniques that are optimized for effectiveness protect resource state, but may do so by severely limiting concurrency of access. As an example, consider the locking of files to preserve consistency in concurrent systems. Pessimistic locking is most effective at preserving state, but results in a complete loss of concurrency whenever the file is locked for writing. Optimistic locking allows a higher degree of concurrency, but may perform worse in a high state of conflict as many transactions must be aborted. In the extreme case of efficient concurrency, conflicts may simply be flagged and left for a separate mechanism to resolve later. This is how inconsistencies are addressed after a disconnection in distributed file systems such as Coda [11]. Similar analogies for access concurrency exist with respect to other resources such as memory protection and process scheduling.

In this emerging class of large distributed systems, the issue is that a high degree of concurrency within a system demands efficiency, while individual users will expect effective consistency preservation. Measures, such as conflict resolution, have not been well explored. It is a non-trivial problem to automatically resolve conflicts on information that does not have a high degree of structure, such as files and ad hoc databases (i.e. the Windows registry). Additionally, there exist a set of resources for whom resolution may not be appropriate after the fact, and large scale active conflict avoidance is a necessity.

2.2 Resource Distribution

Resource distribution describes the degree to which a system has been spread across a network, and how dynamic resources are within it. Even the smallest degree of resource distribution mandates a substantial amount of overhead within a system. Consider the difference between accesses to a local file versus a remote file service such as NFS: both cases contain all of the complexity involved in reading a file from disk, however the remote access has the additional responsibilities of locating the service, marshalling data in and out of message structures, interacting across the network, and handling a considerably larger set of potential error cases.

Transparency, a hallmark goal of distributed systems only obfuscates this problem by concealing the details of distribution. Mechanisms such as remote procedure calls (RPC), which were intended to simplify application development, force distribution to be implemented deep within the system. This results directly in many of the problems traditionally associated with distributed systems such as fragility and inflexibility.

The troubling aspect in this line of consideration is that these issues indicate a fundamental flaw at the very onset of approaches to distribution. RPC really only provides one degree of distribution, by passing a call to a single remote host. With RPC, we have only just entered the arena of distributed systems, and already complexity is overbearing.

Assuming that resources can be accessed in an expressive and reliable manner, a larger problem exists in their distribution. In order to access resources, it must be possible to first locate them. Furthermore, if resources are not static within a system, mechanisms must exist to find them in an ongoing manner. For instance, the location of a resource may have to be determined through a directory service and refreshed with each successive access. In very large scale or highly dynamic systems, a centralized service may not be sufficient to track resource location and other methods, such as forwarding pointers [4], may have to be employed.

2.3 Four Phyla of Applications

From the two axes described above, we draw four phyla of distributed applications, shown in Figure 1. Note that the respective sizes of these domains are by no means equal, we represent this division as it is for simplicity.

What follows is a very brief presentation of each of the four classes. In each case, we supply an example of the phylum to demonstrate its characteristics. We also

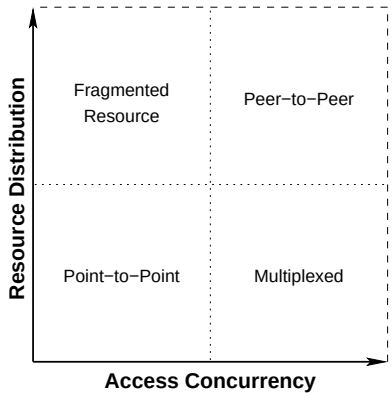

Figure 1: Taxonomy of Distributed Applications

try to identify weaknesses that exist within the domain that may not be acceptable within more advanced systems.

2.3.1 Point-to-point

The point-to-point phylum represents a very simple set of applications in which a client connects to a resource for unshared access. Point-to-point examples exist primarily as components of more complex applications, for instance the data channel of an FTP session is point-to point, in that all of the associated resources are allocated at both ends of the connection at the beginning of a transfer. We would also consider simple RPC to be primarily a point-to-point application, provided that the RPC server handles a single request at a time.

Point-to-point applications are characterized by the fact that the distribution aspects of the system are typically quite visible. As such, when failure does occur it can be identified and resolved primitively by the user. If an FTP server does not respond or crashes during a transfer, the user can attempt a connection somewhere else. Clearly this is not a good system property, however it is generally tolerable within the domain of simple applications.

2.3.2 Multiplexed

Multiplexed applications are those in which resources are delivered with a high degree of concurrency, and possibly conflict control, over a relatively small scale of distribution. File and web servers are excellent examples of this phylum as they often provide a set of centralized resources to large number of concurrent users. Note that in our model, both file and web servers have a high degree of access concurrency, but are still barely distributed. This is because users typically need only connect to a single point to access resources. More distributed examples of a multiplexed applications are distributed striped file systems [18], and scalable data structures [15]. In both of these cases, users may still connect to a single resource, but that resource may forward requests through an additional link to an appropriate secondary server.

The risk of failure is more significant in multiplexed systems because, on the resource provision side, failure has the potential to affect a much larger number of users. To mitigate this problem, very large multiplexed services are often served by specialized hosting facilities where a very high degree of resource reliability may be assumed. Further precautions may involve the installation of redundant resources that take over in the rare case of system failure.

2.3.3 Fragmented Resource

Fragmented systems are those in which resources are spread across, or move within, a set of connected endpoints. Communication is substantially more complex in these systems as messages may not travel directly to a resource, but instead may lead to a cascade of interactions across the system. The domain naming service (DNS) is a well-known example of this type of system, and demonstrates many of the entailing difficulties. For example, the need to protect seperate administrative domains often require updates to be made by hand, resulting in very slow adaptation.

Fragmented resource systems provide the benefit of distributing resources in a broad scope, possibly even providing redundancy. However, fragmenting resources means that administration also becomes divided, which adds an overhead in terms of system administration and maintenance. This property may be an explanation as to why more advanced directory services, such as LDAP, have failed to achieve broad acceptance within the Internet.

2.3.4 Peer-to-peer

Peer-to-peer applications are highly distributed and involve a high degree of potentially conflicting, concurrent access to resources. This is a fairly hypothetical description, as very few such applications currently exist at the Internet scale. Peer-based file sharing applications, such as Gnutella [1] and Freenet [7], are initial steps within this domain but only begin to enter the phylum. Gnutella does not need to address any conflict issues, nor has it proven able to scale.

In this class of application, the acceptable weaknesses within the other phyla compound and cannot be avoided. Failure has a high potential impact, but resources cannot be protected. Administration is distributed and the coupling between administrative domains may become much more dynamic. We discuss these issues more extensively in the next section.

3 Open Architectural Problems

Based on our taxonomy and a survey of existing systems, we identify a set of four prevalent architectural problems that currently inhibit the development of advanced distributed systems. These problems are failure resolution, resource management, administration, and communication infrastructure.

3.1 Failure Resolution

Despite the advanced state of systems research, we are still unable to definitively tell when a resource has failed. Non-terminal failure states, such as livelock and Byzantine failure are incredibly difficult to detect and resolve. Furthermore, in large distributed systems, small failures have the potential to cascade across a system, snowballing towards disaster.

Traditional design goals, such as transparency and layering, force failure to be resolved inappropriately, often requiring that it be masked within a system. General purpose failure handlers cannot predict all possible fail states, and so are unable to effectively address out-of-band failure.

There does not currently exist an accepted, universal approach to expressing, detecting, and resolving failure in distributed systems. Clearly, not all failures can be detected and resolved, but in this situation, it is not clear what systems should do to cope and maintain a degree of sanity.

3.2 Resource Management

In order to carry reliable services beyond the confines of locked facilities, we need to be able to expect the same reliable levels of service from end nodes and connective infrastructure in the distributed environment. Applications desiring a high degree of reliability must be able to reserve resources and comfortably expect that those reservations will be upheld. Unfortuantely, the use of reservation systems such as RSVP [9] presents support for this problem but do not solve it. Reservation schemes inevitably present the possibility of a reduction in available resources, a situation akin to partial failure, to which there is no real analogy in a local high speed network. Tolerating a reduction in service quality, or other sudden change in resource availability requiress a fundamental change in system design.

Furthermore, in a highly distributed environment it is naive to assume that resources will remain available. Applications must be able to gracefully handle resource loss and reallocation. Additional mechanisms, such as redundancy, must be supported within the system to guard against failure.

3.3 Administration

The fragmentation of resources mandates a need to provide adaptable, configurable systems in an environment where control itself is distributed. Models must be developed that allow the scaling of administration in systems with arbitrary (i.e. non-hierarchical) structure. Systems must define and support techniques for allowing a variety of levels of trust in relationships between participants.

It is very likely that a solution to this particular area involves the localization of administration to the highest possible degree. More specifically, individual users and local administrative bodies will be responsible for configuring all aspects of their local systems. However, in distributed systems where resources can potentially be shared with remote, administratively disjoint parties, mechanisms must exist to effectively handle and express changes across administrative boundaries. These mechanisms necessarily must allow the delegation of trust and responsibility in an appropriate manner.

3.4 Communication Infrastructure

Distributed systems are dependent on, and arguably defined by, their communications infrastructure. Although the existing TCP/IP network and overlying network interfaces within operating systems have surpassed all expectations of scalability, they have also remained essentially unchanged for the life of the Internet. The existing network presents many hinderances to advanced distributed systems and several are worth addressing briefly here.

There exists no well-developed infrastructure for group communications. IP multicast, although a substantial improvement to the existing network, has questionable scalability and performance for use in a large and dynamic system and may possess significant vulnerabilities. Non-multicast communication remains inextricably tied to (and identified by) endpoints, making mobility and management difficult.

More importantly, methods of collaboration involving more than two participants are not yet available. Interacting with a set of resources is almost universally handled through a coordinating resource, which typically leads to a single point of failure and congestion within systems. In order for peer-to-peer applications to become a reality, mechanisms that allow groups to work together in efficient ways must be developed.

4 Open Adoptional Problems

Through the use of our taxonomy, we have been able to identify structural issues restraining the development of advanced distributed systems. If all of these issues were to be solved and a system constructed, it would doubtlessly be a considerable research contribution. However, we feel that such a system would inevitably flounder were it to be made available for broad use within the Internet. In this section, we identify a set of open problems that are not identified by our taxonomy. These problems are not defined directly by the structure of a system, but rather are necessary properties for it to be useful in the real world.

4.1 Physical Resource Discovery and Naming

It is incredibly difficult to provide a useful integration between distributed systems and the physical world. Network topologies, especially as exposed by existing protocols, provide an entirely unrepresentative view of resource location. A strongly desired property of advanced distributed systems for ubiquitous [14] and pervasive [12] computing is to allow mobile users to adapt to locally available resources. For instance, it is desirable to easily locate and access a hotel printer. Although much work has emerged in recent years addressing the naming and discovery of resources in a physical dimension [17, 13], the problem has hardly been solved. The emergence of mobile devices that provide geographic information will doubtlessly make this problem even more relevant.

4.2 Security and Privacy

Concerns over privacy and security clearly escalate as resources become more distributed. Centralized, and even lightly distributed systems have proven able to use access control lists (ACLs) and encryption to effectively protect resources. However, as systems (or perhaps administration) become too distributed for centralized solutions, alternate mechanisms must be considered. Capabilities have been touted as a solution within the distributed case that have yet to see a successful broad application. Capabilities have inherent problems with respect to access revocation, which typically requires the rekeying of resources and reauthorizing clients. Furthermore, capabilities are very difficult to administer and track within the context of broad distribution. Finally, as long-lived resources that are protected by encryption, capabilities may be vulnerable to attack.

4.3 Economies of Sharing

A frequently cited benefit to the development of fine-grained distributed systems is the opportunity to share unused resources with others [16, 5]. The reasoning behind this approach is that no one uses all of their resources all of the time, so a low-overhead sharing scheme should be globally beneficial. Gnutella represents a real-world test of this philosophy, in that users are able, but not required, to share local files with others. A study from Xerox PARC [3] shows that users do not behave fairly and that a very few hosts actually share at all. OceanStore [8] proposes a utility-based system for file storage in which resources would be exchanged and billed between administrations in a manner analogous to the power system. Other systems for information sharing [2] on the net have involved artificial economies of karma, that is exchanged between participants.

There is considerable opportunity to explore how sharing should be provided within distributed systems. An effective solution to this problem will have a strong effect on the overall success of these systems. Additionally, in a system where resources such as network bandwidth are shared arbitrarily and perhaps anonymously, there remain questions regarding the payment for these services. As the economy of the Internet is based on the traffic patterns of existing applications, the emergence of a widely adopted system that drastically changes these patterns has the potential to disrupt the financial operation of the net itself.

4.4 System Evolution

Simply providing a large scale system is a considerable feat. The ongoing maintenance and evolution of such a system is considerably more difficult. The Internet is plagued with evolution issues, as systems have not been designed with change in mind. The Hypertext Markup Language (HTML) has evolved through several generations, but authors must still provide backwards-compatibility for legacy browsers at the expense of being able to use new features. The Internet itself is an outstanding example of this problem: the next generation Internet protocol, IPv6, has been in development and limited use for years. The implications of rolling out the protocol across the entire Internet are incredible, and the new protocol provides no easier mechanism for its own inevitable evolution.

Systems must be designed with evolution in mind. Architectural assumptions and application couplings must be minimized wherever possible. Methods must be developed that allow complete systems to be upgraded and changed drastically with a low negative impact on the environment as a whole.

4.5 Heterogeneity

In massive distributed systems, it is not reasonable to expect or mandate uniformity across resources. To do so limits innovation and flexibility and also inhibits evolution, as described above. In order for systems to be flexible and improve over time, the implementation requirements of individual resources must be as light as possible. Furthermore, the requirements must themselves be able to change over time.

4.6 Software Structure

The representation of the network within application code is often an abstract and independant functional unit; client and server source are completely disjoint, obscuring the coupling that is inherent within the system. As stated above, attempts to build systems that transparently handle distribution make it impossible to appropriately expose and resolve failure. However, exposing distribution completely leads to systems whose complexity makes application development considerably more difficult.

Recently, the aspect-oriented programming (AOP) community [6] has focused attention on the concept of *crosscutting concerns*, which are elements of a system that cut through the primary system modularity. They have proposed linguistic mechanisms intended to allow implementation of these concerns as first class modules, called *aspects*. AOP may present the potential to write code that describes functionality across the network, while addressing fault and control issues appropriately. In AOP, we see what may be a new means of gaining the benefits of transparency without the associated weaknesses.

5 Conclusion

The purpose of this paper has been to identify problems that necessarily must be addressed in order to develop advanced, Internet-scale distributed systems. Through a taxonomical observation of existing systems, we have identified a set of open architectural problems including failure resolution, resource management, administration, and communication infrastructure. We then presented a set of six adoptional problems whose solutions will strongly support the acceptance of large distributed applications within the network. Projects to develop environments for ubiquitous [14], invisible [10], and pervasive [12] distributed applications have, and continue to be, very exciting research that will need to address many of these issues in order to realize their visions.

References

[1] Gnutella. http://gnutella.wego.com.

[2] Mojo nation. http://www.mojonation.com/.

[3] E. Adar and B. Huberman. Free riding on Gnutella. Technical report, Xerox PARC, August 2000.

[4] C. Amza and A. Cox. Treadmarks: Shared memory computing on networks of workstations. *IEEE Computer*, Feb 1996.

[5] J. Basney, M. Livny, and T. Tannenbaum. Deploying a high throughput computing cluster. In *High Performance Cluster Computing*. Prentice Hall, 1999.

[6] G. Kiczales et al. Aspect-oriented programming. In *European Conference on Object-Oriented Programming (ECOOP)*, 1997.

[7] I. Clarke et al. Freenet: A distributed anonymous information storage and retrieval system. In *Proc. of the ICSI Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[8] J. Kubiatowicz et. al. Oceanstore: An architecture for global-scale persistant storage. In *ASPLOS*, 2000.

[9] L. Zhang et al. RSVP: A new resource reservation protocol. *IEEE Network Magazine*, September 1993.

[10] M. Esler et al. Next century challenges: Data-centric networking for invisible computing. In *Mobile Computing and Networking*, 1999.

[11] M. Satyanarayanan et al. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4), 1990.

[12] R. Grimm et al. A system architecture for pervasive computing. In *ACM SIGOPS European Workshop*, September 2000.

[13] S. E. Czerwinski et al. An architecture for a secure service discovery service. In *Mobile Computing and Networking*, 1999.

[14] S. Gribble et al. The ninja architecture for robust internet-scale systems and services. In *Special Issue of Computer Networks on Pervasive Computing*, 2000.

[15] S. Gribble et al. Scalable, distributed data structures for internet service construction. In *OSDI*, 2000.

[16] T. E. Anderson et al. Serverless network file systems. *ACM Transactions on Computer Systems*, 14(1), February 1996.

[17] W. Adjie-Winoto et al. The design and implementation of an intentional naming system. In *SOSP*, 1999.

[18] J. Hartman and J. Ousterhout. The Zebra striped network file system. *ACM Transactions on Computer Systems*, 13(3), 1995.