Back to the Future: dependable computing = dependable services

Jeffrey Chase, Amin Vahdat*, and John Wilkes

Abstract

Clients are coming to rely more and more on external services to meet the needs of their users, and the clients are increasingly simple caches of soft state – "truth" is maintained elsewhere. As a result, the user experience of dependability is better served by making those services ultra-dependable than by increasing the reliability of an individual client. We explore here some of the consequences of this statement, and conclude that developing scalable, dependable services may be a more fruitful approach than an extreme emphasis on "dependable OSes". Along the way we look at quantifying "dependability" in this new world; some of what it takes to provide dependable, large-scale services; and some approaches that we are exploring to do so.

1 Back to the Future

Once upon a time, the US space program funded the development of ultra-dependable writing instruments. The result was an astounding piece of technology: a pen that could write upside down, in zero gravity, underwater, etc. It was priced accordingly. NASA's extreme conditions required it. But the rest of us simply pick up a new, cheap pen when the one we are using stops working, and think nothing of discarding the old one. We find it more economical and convenient to rely on a service that supplies pens, not on a super-dependable pen.

The analogy above hints at our position on this topic: user perception of "dependability" is increasingly driven by the dependability of the underlying services rather than by the dependability of an individual client. Why is this?

Increasingly, clients are I/O devices rather than computing platforms. They are becoming more diverse, cheaper, and more specialized. And they are all connected to the service infrastructure. Any device that captures data (cameras, laptops, sensors, phones) has to preserve the data for later access by other services and applications. Any device that presents data to a user (MP3 players, viewers, WAP phones, monitors) gets it from its environment.

The trend is driven by Moore's Law and better connectivity: rather than simply becoming more powerful, client devices are trading some of that power for more portability. Further, they are increasingly communication-oriented devices rather than pure computing devices – which depends on connectivity, but connectivity is not the barrier to dependability that we once thought it was. In our homes and offices, broadband is as dependable as our phone and electrical systems, which we take for granted. On the road, we have good coverage from cellular phone networks and 802.11 in public places. And communication performance will improve dramatically as we pave over the "last mile." At the same time, clients are acquiring increasing abilities to cache data, so temporary disconnections are less problematic.

Meanwhile, applications are increasingly server-based. There are several drivers for this, including:

1. Many services are intrinsically based on shared state (banking, commerce, trading, reservations, google, yahoo) or communication through resilient (albeit temporary) state (e.g., mail, messaging).

2. Growing numbers of services provide a place to upload data and share it with other people or with other devices owned by the same user.

3. Shared information is of intrinsically higher value than information that is usable only by one person; and that value is in proportion to the number of people who come together to take advantage of it.

4. We are becoming used to higher levels of dependability – we treat it as an unwarranted exception when an airline booking system doesn't work, rather than the small miracle that it is when things go well.

5. The very portability of clients means that they are subjected to a greater range of threats than the fixed computing systems of yore: a device that can easily get dropped, stolen, or lost is not an ideal environment for preserving the only copy of valuable, long-term state.

*Contact Author: Amin Vahdat, Box 90129, Department of Computer Science, Durham NC 27708, *vahdat@cs.duke.edu*

We believe that the computing environment will evolve to one in which anything involving storage will be backed by reliable servers in controlled environments – and client devices will become ever more interchangeable, merely display devices and caches for data and software. This will bring its own challenges, such as consistency issues – clients will temporarily store data in write-back caches when they become disconnected, pending transmission to the service. Dealing with this will be an important service-architecture question – but all our experience with distributed file systems suggests that it will be manageable.

Consider the oft-used metaphor of the electricity supply, with a small twist. Client devices run on batteries, which are just a cache for electricity generated by the utility and delivered through the wall. Making the batteries more reliable and higher capacity does improve the quality of use, but ultimately we depend on that electric utility being there for us to do our work. But note that it's not a particular power source in that utility that we depend on: individual users typically do not care which site actually generated the electricity, they just want the power. Similarly, when users access a service, they care less and less about where it runs.

This is not a new idea: thin clients, network computers, and now scalable utility computing. But it is happening just the same, and participation from the operating system community is central to achieving the vision if we are to meet the levels of dependability that people are coming to take for granted across an ever-wider range of services.

2 Utility computing as the path to dependability

We believe that server-based computing and self-organizing resource utilities (server/network/storage farms) are the basis for dependable computing in the future. What will it take to realize this? At some level, much of it is simply good resource management, coupled with development of appropriate resource and service abstractions. A few things complicate this: the sheer scale (millions of clients, not tens or hundreds); the rapid rate of change of demand levels, enabled by the any-to-any connectivity offered by the Internet; the economics of supporting a utility-based infrastructure; and all the privacy, data integrity, security, and service-level predictability demands that "dependability" implies. We need to extend the Internet reliability and robustness model to services: we want to detect failures and route around them, as transparently to end users as possible.

Building robust services today requires cluster-based techniques where potentially thousands of individual machines deliver some higher level service (e.g., google's web search scheme). Similarly, geographic replication and transparent request redirection (e.g., using Akamai DNS servers) are employed to avoid network congestion and individual failures. All these techniques are transparent to end users who simply request a service and are agnostic as to who actually delivers it. Such separation of service from a specific machine offers the promise of eliminating Lamport's Pitfall, where "A distributed system is one in which the failure of a machine I've never heard of can prevent me from getting my work done."

This naturally leads to solutions that enable services to be dynamically provisioned – and then to dynamic resource provisioning for network, computational resources, storage, memory, etc. Furthermore, we look to schemes that allow business-driven levels of performance and dependability to be specified – and followed. To support this, we are increasingly able to provision sufficient resources "on demand", quickly enough to deliver desired service levels under rapidly-changing loads.

This has been accomplished, in part, by better understanding of service-level agreements (SLAs). SLAs used in computer networks have demonstrated the benefits of using economic incentives to ensure well-provisioned services. That is, availability of sufficient resources is much more likely if delivering better performance and dependability results in more revenue.

Fundamental to our approach are the following techniques:

- The use of SLAs, both to quantify the desired goals, and to provide economic incentives for the utility providers.

  We hope that providing cost models for resources will motivate application developers to deploy efficient software for a given demand level. Even if this is not the case, similar models can be provided at the resource-management layer.

- Mechanisms to allow the resource utility to provision to deliver target levels of performance and reliability. This includes mechanisms to prioritize resource allocations during temporary overload.

- Simultaneously performing replica placement, resource routing, and overlay topology configuration to achieve target levels of "performance" for minimal "cost".

- Scalable algorithms for maintaining the utility through the aggressive use of caching, approximate information, hierarchy, and aggregation.

- Achieving robustness by deploying additional resources and redundancy. There are many examples of this principle, and they make server-computing inherently more dependable: RAID, dynamic replication, redundant paths, multipath routing, session recovery, edge caching and stashing, dynamic service placement and migration.

- Making all these techniques self-managing, so that people do not have to be involved in the systems' response to events (load changes, failures, etc.).

The above list applies mostly at the resource layer. It is also fruitful to consider application-level adaptations: ideally, they should be structured to be fluid, i.e., independent of the number and placement of servers and how load is divided among them. Applications should allow the system infrastructure (utility) to determine service placement, replication degree, and binding to peer services (databases, file servers) in a multi-tier structure. In this way, the utility can monitor conditions, adapt to failure, dynamically adjust placement and redundancy degree, scale up or scale back, and (re)allocate available resources to provide the best global service (for application-specific definitions of "best").

Between these two levels are frameworks that provide for application deployment, and adaptation to resource or application failures that can be accommodated by reassignment of resources to a service, and rebooting [6].

3 Examples of service utilities

3.1 Opus

Opus [2] is an overlay peer utility service. It allows individual applications to specify their performance and availability requirements. Based on this information, Opus initially maps applications to individual nodes across the wide area. Once this has been done, observed access patterns to individual applications are used to dynamically reallocate resources to match application requirements. For example, if many accesses are observed for an application in a given network region, Opus may reallocate additional resources close to that location.

One key challenge to achieving this model is determining the relative utility of a given candidate configuration. That is, for each available unit of resource, we must be able to *predict* how much any given application would benefit from that resource. Existing work in resource allocation in clusters [3] and replica placement for availability [10] indicate that this can be done efficiently in a variety of cases.

One key aspect of our work is the use of Service Level Agreements (*SLAs*) to specify the amount each application is willing to "pay" for a given level of performance. Opus uses utility functions for this: it makes allocation and deallocation decisions based on the expected relative benefit of a set of target configurations, based on an estimate of the marginal utility of resources across a set of applications at current levels of global demand [3].

Opus employs a global *service overlay* to maintain soft state about the current mapping of utility nodes to hosted applications (group membership). This service overlay is key to many individual system components, such as routing requests from individual clients to appropriate replicas, and performing resource allocation among competing applications. Individual services running on Opus employ *per-application overlays* to disseminate their own service data and metadata among individual replica sites.

Clearly, a primary concern is ensuring the scalability and reliability of the service overlay. Opus addresses this through the aggressive use of hierarchy, aggregation, and approximation in creating and maintaining scalable overlay structures.

3.2 The Grid

Although it initially began as a way for scientific applications to use "excess" computing cycles at other institutions, the proponents of *The Grid* have recently embraced a more general model for resource management and sharing across a federated set of suppliers, and recent work on defining an "open grid service architecture" [5] has made it clear that the eventual target is no longer limited to relatively short-lived jobs, but also embraces longer-lived services.

3.3 Planetary scale computing

Beginning with the HP Utility Data Center [4], a product to enable the deployment of a first form of managed utility computing, HP has entered on a path to develop technology to enable what they call "planetary scale computing," or "service-centric computing" – essentially the vision espoused here. Here, the data center runs a "utility OS" [8], whose dependability is crucial to the availability of services that the data center supports. Such an "OS" has to deal with all the usual issues: resource management, provision of abstractions, client isolation ... except that the resources are entire processor nodes, or portions of disk arrays, and shared networking infrastructure, rather than the more traditional memory pages, CPUs, and IO cards.

Existing HP research work on automatic management of storage system services has demonstrated that the "lights out" provisioning of resources to meet application needs is a viable approach [1]; the next step is to apply these ideas to the broader scope of the entire data center.

4 Defining dependability

Implicit in this whole discussion is an underlying notion of what "dependability" means. Today's storage vendors and web server hosting services often use percentage uptime (e.g., 99.99%) to describe system dependability. This is a simple availability metric – "is it up?" – which, although somewhat useful for a single computer, such as a client, is inadequate when the larger context is considered,

because failures often degrade service rather than fully interrupt it.

A better notion is *performability*, which we define to mean "what portion of the time is the system meeting [the user's] expected service levels?" Given such a definition, we can start to judge alternative service designs and offerings, and then go on to design a service deployment against its user expectations.

A service is useful only if a user's requests can be processed within their *tolerance*, or expectation. The tolerance can include a rich combination of aspects, including throughput, latency, accuracy, completeness, and consistency (e.g., the service may return "slightly" inconsistent [9] data in exchange for improved overall accessibility).

Inadequate performance may result from many causes: network congestion, server overload, partial failures of resources, or partial data inaccessibility (or even loss); or even simply stringent user expectations. A service may be "unavailable" from a particular user's perspective even when the system is up and running - and this is a particular problem during times of peak demand, which are precisely the times when the system needs to be most dependable.

Interruptions may be frequent and short, or rare and long. Do these have the same "average" dependability? This depends on what the user expectation is. For example, if the interruptions are frequent enough to prevent them completing a transaction, then they are unlikely to be satisfied, whatever the "average" may indicate.

Our approach to building dependable systems has applications specifying the relative value ("utility") of various levels of performability and data consistency. A specific example of this kind of service specification for the storage-systems space can be found in [7]. Explicit in this proposal is the notion that there may be more than one appropriate service level, and that the traditional "all or nothing" distinction may not be sufficient – "is it an acceptable service?" is a more sophisticated question than "is it up?".

In this manner, the compute utility can determine how to provision available resources to maximize per-service dependability in the face of individual failures, changing network conditions, and dynamic client access patterns.

5 Conclusions

This paper takes somewhat of a contrarian position on the question of how to build a dependable operating system. We believe that the traditional operating system, defined as a monolithic structure mediating all application access to host software, is becoming less and less important as a determiner of dependability. Rather, the "operating system" is being extended to cover the gamut of management and deployment issues involved in executing an entire service, across the network [8].

Thus, we believe that the operating system research agenda must address issues that encompass the concerns raised by such global scale resource-management: how should the "operating system" best manage global network resources to deliver reliable services transparently to millions of simultaneous users? How should it dynamically place functionality and employ redundancy to deliver much better performance and availability than any centralized host or single client system could? Dependable computing is not (just) about building a more robust UNIX or Windows. Rather, it is about thin, stateless, disposable clients utilizing dependable communication to access global, dependable, service utilities.

References

[1] Eric Anderson, Michael Hobbs, Kimberly Keeton, Susan Spence, Mustafa Uysal, and Alistair Veitch. Hippodrome: Running Circles Around Storage Administration. In *Conference on File and Storage Technology (FAST'02)*, January 2002.

[2] Rebecca Braynard, Dejan Kostić, Adolfo Rodriguez, Jeffrey Chase, and Amin Vahdat. Opus: an Overlay Peer Utility Service. In *Proceedings of the 5th International Conference on Open Architectures and Network Programming (OPENARCH)*, June 2002.

[3] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. Managing Energy and Server Resources in Hosting Centers. In *Proceedings of the 18th ACM Symposium on Operating System Principles (SOSP)*, October 2001.

[4] Hewlett Packard Corporation. Utility Data Center. `www.hp.com/solutions1/infrastructure/solutions/utilitydata/overview/`, 2001.

[5] Ian Foster, Carl Kesselman, Jeffrey Nick, and Steven Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, January 2002.

[6] Patrick Goldsack. SmartFrog: a framework for configuration. In *Large Scale System Configuration Workshop*, November 2001.

[7] John Wilkes. Traveling to Rome: QoS Specifications for Automated Storage System Management. In *International Workshop on Quality of Service (IWQoS'2001)*, June 2001.

[8] John Wilkes, Patrick Goldsack, G. (John) Janakiraman, Lance Russell, Sharad Singhal, and Andrew Thomas. eOS - The Dawn of the Resource Economy. Technical report, HP Laboratories, May 2001. Available from `http://www.hpl.hp.com/SSP/papers`.

[9] Haifeng Yu and Amin Vahdat. Design and Evaluation of a Continuous Consistency Model for Replicated Services. In *Proceedings of Operating Systems Design and Implementation (OSDI)*, October 2000.

[10] Haifeng Yu and Amin Vahdat. Minimal Replication Cost for Availability. Technical report, Duke University, January 2002. Submitted for publication.