TOWARDS A CLOUD COMPUTING RESEARCH AGENDA

Ken Birman, Gregory Chockler, Robbert van Renesse

Abstract

The 2008 LADIS workshop on Large Scale Distributed Systems brought together leaders from the commercial cloud computing community with researchers working on a variety of topics in distributed computing. The dialog yielded some surprises: some hot research topics seem to be of limited near-term importance to the cloud builders, while some of their practical challenges seem to pose new questions to us as systems researchers. This brief note summarizes our impressions.

Workshop Background

LADIS is an annual workshop focusing on the state of the art in distributed systems. The workshops are by invitation, with the organizing committee setting the agenda. In 2008, the committee included ourselves, Eliezer Dekel, Paul Dantzig, Danny Dolev, and Mike Spreitzer. The workshop website, at http://www.cs.cornell.edu/projects/ladis2008/, includes the detailed agenda, white papers, and slide sets [23]; proceedings are available electronically from the ACM Portal web site [22].

LADIS 2008 Topic

The 2008 LADIS topic was Cloud Computing, and more specifically:

- Management infrastructure tools (examples would include Chubby [4], Zookeeper [28], Paxos [24], [20], Boxwood [25], Group Membership Services, Distributed Registries, Byzantine State Machine Replication [6], etc),
- Scalable data sharing and event notification (examples include Pub-Sub platforms, Multicast [35], Gossip [34], Group Communication [8], DSM solutions like Sinfonia [1], etc),
- Network-Level and other resource-managed technologies (Virtualization and Consolidation, Resource Allocation, Load Balancing, Resource Placement, Routing, Scheduling, etc),
- Aggregation, Monitoring (Astrolabe [33], SDIMS [36], Tivoli, Reputation).

In 2008, LADIS had three keynote speakers, one of whom shared his speaking slot with a colleague:

- Jerry Cuomo, IBM Fellow, VP, and CTO for IBM's Websphere product line. Websphere is IBMs flagship product in the web services space, and consists of a scalable platform for deploying and managing demanding web services applications. Cuomo has been a key player in the effort since its inception.
- James Hamilton, at that time a leader within Microsoft's new Cloud Computing Initiative. Hamilton came to the area from a career spent designing and deploying scalable database systems and clustered data management platforms, first at Oracle and then at Microsoft. (Subsequent to LADIS, he joined Amazon.com.)
- Franco Travostino and Randy Shoup, who lead eBay's architecture and scalability effort. Both had long histories in the parallel database arena before joining eBay and both participated in eBay's scale-out from early in that company's launch.

We won't try and summarize the three talks (slide sets for all of them are online at the LADIS web site, and additional materials such as blogs and videotaped talks at [18], [29]). Rather, we want to focus on three insights we gained by comparing the perspectives articulated in the keynote talks with the cloud computing perspective represented by our research speakers:

- We were forced to revise our "definition" of cloud computing.
- The keynote speakers seemingly discouraged work on some currently hot research topics.
- Conversely, they left us thinking about a number of questions that seem new to us.

Cloud Computing Defined

Not everyone agrees on the meaning of cloud computing. Broadly, the term has an "outward looking" and an "inward looking" face. From the perspective of a client outside the cloud, one could cite the Wikipedia definition:

> *Cloud computing is Internet (cloud) based development and use of computer technology (computing), whereby dynamically scalable and often virtualized resources are provided as a service over the Internet. Users need not have knowledge of, expertise in, or control over the technology infrastructure "in the cloud" that supports them.*

The definition is broad enough to cover everything from web search to photo sharing to social networking. Perhaps the key point is simply that cloud computing resources should be accessible by the end user anytime, anywhere, and from any platform (be it a cell phone, mobile computing platform or desktop).

The outward facing side of cloud computing has a growing set of associated standards. By and large:

- Cloud resources are accessed from browsers, "minibrowsers" running JavaScript/AJAX or similar code, or at a program level using web services standards. For example, many cloud platforms employ SOAP as a request encoding standard, and HTTP as the preferred way to actually transmit the SOAP request to the cloud platform, and to receive a reply.
- Although the client thinks of the cloud as a single entity, the implementation typically requires one or more data centers, composed of potentially huge numbers of service instances running on a large amount of hardware. Inexpensive commodity PCs structured into clusters are popular. A typical data center has an outward facing bank of servers with which client systems interact directly. Cloud systems implement a variety of DNS and load-balancing/routing mechanisms to control the routing of client requests to actual servers.
- The external servers, which often run in a "demilitarized zone" (outside any firewall), perform "business logic." This typically involves extracting the client request and parallelizing it within some set of services that do the work. The server then collects replies, combines them into a single "result," and sends it back to the client.

There is also an inside facing perspective:

- A cloud service is implemented by some sort of pool of servers that either share a database subsystem or replicate data [14]. The replication technology is very often supported by some form of scalable, high-speed update propagation technology, such as publish-subscribe message bus (in web services, the term Enterprise Service Bus or ESB is a catch-all for such mechanisms).

- Cloud platforms are highly automated: management of these server pools (including such tasks as launching servers, shutting them down, load balancing, failure detection and handling) are performed by standardized infrastructure mechanisms.
- A cloud system will often provide its servers with some form of shared global file system, or in-memory store services. For example, Google's GFS [16], Yahoo!'s HDFS [3], Amazon.com's S3 [2], memcached [26], and Amazon Dynamo [13] are widely cited. These are specific solutions; the more general statement is simply that servers share files, databases, and other forms of content.
- Server pools often need ways to coordinate when shared configuration or other shared state is updated. In support of this many cloud systems provide some form of locking or atomic multicast mechanism with strong properties [4], [28]. Some very large-scale services use tools like Distributed Hash Tables (DHTs) to rapidly find information shared within a pool of servers, or even as part of a workload partitioning scheme (for example, Amazon's shopping-cart service uses a DHT to spread the shopping cart function over a potentially huge number of server machines).

We've focused the above list on the interactive side of a data center, which supports the clustered server pools with which clients actually interact. But these in turn will often depend upon "back office" functionality: activities that run in the background and prepare information that will be used by the servers actually handling client requests. At Google, these back office roles include computing search indices. Examples of widely known back-office supporting technologies include:

- Scheduling mechanisms that assign tasks to machines, but more broadly, play the role of provisioning the data center as a whole. As we'll see below, this aspect of cloud computing is of growing importance because of its organic connection to power consumption: both to spin disks and run machines, but also because active machines produce heat and demand cooling. Scheduling, it turns out, comes down to "deciding how to spend money."
- Storage systems that include not just the global file system but also scalable database systems and other scalable transactional subsystems and middleware such as Google's BigTable [7], which provides an extensive (conceptually unlimited) table structure implemented over GFS.
- Control systems for large-scale distributed data processing like MapReduce [11] and DryadLINQ [37].
- Archival data organization tools, applications that compress information or compute indexes, applications that look for duplicate versions of objects, etc.

In summary, cloud computing lacks any crisp or simple definition. Trade publications focus on cloud computing as a realization of a form of ubiquitous computing and storage, in which such functionality can be viewed as a new form of cyber-supported "utility". One often reads about the cloud as an analog of the electric power outlet or the Internet itself. From this perspective, the cloud is defined not by the way it was constructed, but rather by the behavior it offers. Technologists, in turn, have a tendency to talk about the components of a cloud (like GFS, BigTable, Chubby) but doing so can lose track of the context in which those components will be used – a context that is often very peculiar when compared with general enterprise computing systems.

Is the Distributed Systems Research Agenda Relevant?

We would like to explore this last point in greater detail. If the public perception of the cloud is largely oblivious to the implementation of the associated data centers, the research community can seem oblivious to the way mechanisms are used. Researchers are often unaware that cloud systems have overarching design principles that guide developers towards a cloud-computing mindset quite distinct from what we may have been familiar with from our work in the past, for example on traditional client-server systems or traditional multicast protocols. Failing to keep the broader principles in mind can have the effect of overemphasizing certain cloud computing components or technologies, while losing track of the way that the cloud uses those components and technologies. Of course if the use was arbitrary or similar enough to those older styles of client-server system, this wouldn't matter. But because the cloud demands obedience to those overarching design goals, what might normally seem like mere application-level detail instead turns out to be dominant and to have all sorts of lower level implications.

Just as one could criticize the external perspective ("ubiquitous computing") as an oversimplification, LADIS helped us appreciate that when the research perspective overlooks the roles of our technologies, we can sometimes wander off on tangents by proposing "new and improved" solutions to problems that actually run contrary to the overarching spirit of the cloud mechanisms that will use these technologies.

To see how this can matter, consider the notion of distributed systems consistency. The research community thinks of consistency in terms of very carefully specified models such as the transactional database model, atomic broadcast, Consensus, etc. We tend to reason along the following lines: Google uses Chubby (a locking service) and Chubby uses State Machine Replication based on Paxos. Thus Consensus, an essential component of State Machine Replication, should be seen as a legitimate cloud computing topic: Consensus is "relevant" by virtue of its practical application to a major cloud computing infrastructure. We then generalize: research on Consensus, new Consensus protocols and tools, alternatives to Consensus are all "cloud computing topics".

While all of this is true, our point is that Consensus, for Google, wasn't the goal. Sure, locking matters in Google, this is why they built a locking service. But the bigger point is that even though large data centers need locking services, if one can trust our keynote speakers, application developers are under huge pressure *not to use them.* We're reminded of the old story of the blind men touching the elephant. When we reason that "Google needed Chubby, so Consensus as used to support locking is a key cloud computing technology," we actually skip past the actual design principle and jump directly to the details: this way of building a locking service versus that one. In doing so, we lose track of the broader principle, which is that distributed locking is a bad thing that must be avoided!

This particular example is a good one because, as we'll see shortly, if there was a single overarching theme within the keynote talks, it turns out to be that strong synchronization of the sort provided by a locking service must be avoided like the plague. This doesn't diminish the need for a tool like Chubby; when locking actually can't be avoided, one wants a reliable, standard, provably correct solution. Yet it does emphasize the sense in which what we as researchers might have thought of as the main point ("the vital role of consistency and Consensus") is actually secondary in a cloud setting. Seen in this light, one realizes that while research on Consensus remains valuable, it was a mistake to portray it as if it was research on the most important aspect of cloud computing.

Our keynote speakers made it clear that in focusing overly narrowly, the research community often misses the bigger point. This is ironic: most of the researchers who attended LADIS are the sorts of people who teach their students to distinguish a problem statement from a solution to that problem, and yet by overlooking the *reasons* that cloud platforms need various mechanisms, we seem to be guilty of fine-tuning specific solutions without adequately thinking about the context in which they are used and the real needs to which they respond – aspects that can completely reshape a problem statement. To go back to Chubby: once one realizes that locking is a technology of last resort, while building a great locking service is clearly the right thing to do, one should also ask what research questions are posed by the need to support applications *that can safely avoid locking*. Sure, Consensus really matters, but if we focus too strongly on it, we risk losing track of its limited importance in the bigger picture.

Let's look at a second example just to make sure this point is clear. During his LADIS keynote, Microsoft's James Hamilton commented that for reasons of autonomous control, large data centers have adopted a standard model resembling the well-known Recovery-Oriented Computing (ROC) paradigm [27], [5]. In this model, every application must be designed with a form of automatic fault handling mechanism. In short, this mechanism *suspects* an application if any other component complains that it is misbehaving. Once suspected by a few components, or suspected strenuously by even a single component, the offending application is rebooted – with no attempt to warn its clients or ensure that the reboot will be graceful or transparent or non-disruptive. The focus apparently is on speed: just push the reboot button. If this doesn't clear the problem, James described a series of next steps: the application might be automatically reinstalled on a fresh operating system instance, or even moved to some other node—again, without the slightest effort to warn clients.

What do the clients do? Well, they are forced to accept that services behave this way, and developers code around the behavior. They try and use idempotent operations, or implement ways to resynchronize with a server when a connection is abruptly broken.

Against this backdrop, Hamilton pointed to the body of research on transparent task migration: technology for moving a running application from one node to another without disrupting the application or its clients. His point? Not that the work in question isn't good, hard, or publishable. But simply that cloud computing systems don't need such a mechanism: if a client can (somehow) tolerate a service being abruptly restarted, reimaged or migrated, there is no obvious value to adding "transparent online migration" to the menu of options. Hamilton sees this as analogous to the end-to-end argument: if a low level mechanism won't simplify the higher level things that use it, how can one justify the complexity and cost of the low level tool?

Earlier, we noted that although it wasn't really our intention when we organized LADIS 2008, Byzantine Consensus turned out to be a hot topic. It was treated, at least in passing, by surprisingly many LADIS researchers in their white papers and talks. Clearly, our research community is not only interested in Byzantine Consensus, but also perceives Byzantine fault tolerance to be of value in cloud settings.

What about our keynote speakers? Well, the quick answer is that they seemed relatively uninterested in Consensus, let alone Byzantine Consensus. One could imagine many possible explanations. For example, some industry researchers might be unaware of the Consensus problem and associated theory. Such a person might plausibly become interested once they learn more about the importance of the problem. Yet this turns out not to be the case for our four keynote speakers, all of whom have surprisingly academic backgrounds, and any of whom could deliver a nuanced lecture on the state of the art in fault-tolerance.

The underlying issue was quite the opposite: the speakers believe themselves to understand something *we* didn't understand. They had no issue with Byzantine Consensus, but it just isn't a primary question for them. We can restate this relative to Chubby. One of the LADIS attendees commented at some point that Byzantine Consensus could be used to improve Chubby, making it tolerant of faults that could disrupt it as currently implemented. But for our keynote speakers, enhancing Chubby to tolerate such faults turns out to be of purely academic interest. The bigger – the overarching – challenge is to find ways of transforming services that might seem to need locking into versions that are loosely coupled and can operate correctly without locking [18] – to get Chubby (and here we're picking on Chubby: the same goes for *any* synchronization protocol) off the critical path.

The principle in question was most clearly expressed by Randy Shoup, who presented the eBay system as an evolution that started with a massive parallel database, but then diverged from the traditional database model over time. As Shoup explained, to scale out, eBay services started with the steps urged by Jim Gray in his famous essay on terminology for scalable systems [14]: they partitioned the enterprise into multiple disjoint subsystems, and then used small clusters to parallelize the handling of requests within these. But this wasn't enough, Shoup argued, and eventually eBay departed from the transactional ACID properties entirely, moving towards a decentralized convergence behavior in which server nodes are (as much as possible) maintained in loosely consistent but transiently divergent states, from which they will converge back towards a consistent state over time.

Shoup argued, in effect, that scalability and robustness in cloud settings arises not from tight synchronization and fault-tolerance of the ACID type, but rather from loose synchronization and self-healing convergence mechanisms.

Shoup was far from the only speaker to make this point. Hamilton, for example, commented that when a Microsoft cloud computing group wants to use a strong consistency property in a service… his executive team had the policy of sending that group home to find some other way to build the service. As he explained it, one can't always completely eliminate strong ACID-style consistency properties, but the first principle of successful scalability is to batter the consistency mechanisms down to a minimum, move them off the critical path, hide them in a rarely visited corner of the system, and then make it as hard as possible for application developers to get permission to use them. As he said this, Shoup beamed: he has the same role at eBay.

The LADIS audience didn't take these "fighting words" passively. Alvisi and Guerraoui both pointed out that Byzantine fault-tolerance protocols are more and more scalable and more and more practical, citing work to optimize these protocols for high load, sustained transaction streams, and to create optimistic variants that will terminate early if an execution experiences no faults [10], [21]. Yet the keynote speakers pushed back, reiterating their points. Shoup, for example, noted that much the same can be said of modern transaction protocols: they too scale well, can sustain extremely high transaction rates, and are more and more optimized for typical execution scenarios. Indeed, these are just the kinds of protocols on which eBay depended in its early days, and that Hamilton "cut his teeth" developing at Oracle and then as a technical leader of the Microsoft SQL server team. But for Shoup *performance isn't the reason that eBay avoids these mechanisms.* His worry is that no matter how fast the protocol, it can still cause problems.

This is a surprising insight: for our research community, the prevailing assumption has been that Byzantine Protocols would be used pervasively if only people understood that they no longer need to be performance limiting bottlenecks. But Shoup's point is that eBay avoids them for a different reason. His worry involves what could be characterized as "spooking correlations" and "self synchronization". In effect, any mechanism capable of "coupling" the behavior of multiple nodes even loosely would increase the risk that the whole data center might begin to thrash.

Shoup related stories about the huge effort that eBay invested to eliminate convoy effects, in which large parts of a system go idle waiting for some small number of backlogged nodes to work their way through a seemingly endless traffic jam. Then he spoke of feedback oscillations of all kinds: multicast storms, chaotic load fluctuations, thrashing. And from this, he reiterated, eBay had learned the hard way that *any* form of synchronization must be limited to small sets of nodes and used rarely.

In fact, the three of us are aware of this phenomenon from projects on which we've collaborated over the years. We know of many episodes in which data center operators have found their large-scale systems debilitated by internal multicast "storms" associated with publish-subscribe products that destabilized on a very large scale, ultimately solving those problems by legislating that UDP multicast would not be used as a transport. The connection? Multicast storms are another form of self-synchronizing, destructive behavior that can arise when coordinated actions (in this case, loss recovery for a reliable multicast protocol) are unleashed on a large scale.

Thus for our keynote speakers, "fear of synchronization" was an overarching consideration that in their eyes, mattered far more than the theoretical peak performance of such-and-such an atomic multicast or Consensus protocol, Byzantine-tolerant or not. In effect, the question that mattered wasn't actually performance, but rather the risk of destabilization that even using mechanisms such as these introduces.

Reflecting on these comments, which were echoed by Cuomo and Hamilton in other contexts, we find ourselves back in that room with the elephant. Perhaps as researchers focused on the performance and scalability of multicast protocols, or Consensus, or publish-subscribe, we're in the position of mistaking the tail of the beast for the critter itself. Our LADIS keynote speakers weren't naïve about the properties of the kinds of protocols on which we work. If anything, we're the ones being naïve, about the setting in which those protocols are used.

To our cloud operators, the overarching goal is scalability, and they've painfully learned one overarching principle of scale: decoupling. The key is to enable nodes to quietly go about their work, asynchronously receiving streams of updates from the back-office systems, synchronously handling client requests, and avoiding even the most minor attempt to interact with, coordinate with, agree with or synchronize with other nodes. However simple or fast a consistency mechanism might be, they still view such mechanisms as potential threats to this core principle of decoupled behavior. And thus their insistency on asynchronous convergence as an alternative to stronger consistency: yes, over time, one wants nodes to be consistent. But putting consistency ahead of decoupling is, they emphasized, just wrong.

Towards a Cloud Computing Research Agenda

Our workshop may have served to deconstruct some aspects of the traditional research agenda, but it also left us with elements of a new agenda – and one not necessarily less exciting than the one we are being urged by these leaders to shift away from. Some of the main research themes that emerge are:

1. Power management. Hamilton was particularly emphatic on this topic, arguing that a ten-fold reduction in the power needs of data centers may be possible if we can simply learn to build systems that are optimized with power management as their primary goal, and that this savings opportunity may be the most exciting way to have impact today [15]. Examples of ideas that Hamilton floated were:
   - Explore ways to simply do less during surge load periods.
   - Explore ways to migrate work in time. The point here was that load on modern cloud platforms is very cyclical, with infrequent peaks and deep valleys. It turns out that the need to provide acceptable quality of service during the peaks inflates costs continuously: even valley time is made more expensive by the need to own a power supply able to handle the peaks, a number of nodes adequate to handle surge loads, a network provisioned for worst-case demand, etc. Hamilton suggested that rather than think about task migration for fault-tolerance (a topic mentioned above), we should be thinking about task decomposition with the goal of moving work from peak to trough. Hamilton's point was that in a heavily loaded data center coping with a 95% peak load, surprisingly little is really known about the actual tasks being performed. As in any system, a few tasks probably represent the main load, so one could plausibly learn a great deal – perhaps even automatically. Having done this, one could attack those worst-case offenders. Maybe they can precompute some data, or defer some work to be finished up later, when the surge has ended. The potential seems to be very great, and the topic largely unexplored.
   - Even during surge loads, some machines turn out to be very lightly loaded. Hamilton argued that if one owns a node, it should do its share of the work. This argues for migrating portions of some tasks in space: breaking overloaded services into smaller components that can operate in parallel and be shifted around to balance load on the overall data center. Here, Hamilton observed that we lack software engineering solutions aimed at making it easy for the data center development team to delay these decisions until late in the game. After all, when building an application it may not be at all clear that, three years down the road, the application will account for most of the workload during surge loads that in turn account for most of the cost of the data center. Thus, long after an application is built, one needs ways to restructure it with power management as a goal.
2. New models and protocols for convergent consistency [32]. As noted earlier, Shoup energetically argued against traditional consistency mechanisms related to the ACID properties, and grouped Consensus into this technology area. But it was not so clear to us what alternative eBay would prefer, and in fact we see this as a research opportunity.
   - We need to either adapt existing models for convergent behavior (self-stabilization, perhaps, or the forms of probabilistic convergence used in some gossip protocols) to create a formal model that could capture the desired behavior of loosely coupled systems. Such a model would let us replace "loose consistency" with strong statements about precisely when a system is indeed loosely consistent, and when it is merely broken!
   - We need a proof methodology and metrics for comparison, so that when distinct teams solve this new problem statement, we can convince ourselves that the solutions really work and compare their costs, performance, scalability and other properties.
   - Conversely, the Byzantine Consensus community has value on the table that one would not wish to sweep to the floor. Consider the recent, highly publicized, Amazon.com outage in which that company's S3 storage system was disabled for much of a day when a

corrupted value slipped into a gossip-based subsystem and was then hard to eliminate without fully restarting the subsystem – one needed by much of Amazon, and hence a step that forced Amazon to basically shut down and restart. The Byzantine community would be justified, we think, in arguing that this example illustrates not just a weakness in loose consistency, but also a danger associated with working in a model that has never been rigorously specified. It seems entirely feasible to import ideas from Byzantine Consensus into a world of loose consistency; indeed, one can imagine a system that achieves "eventual Byzantine Consensus." One of the papers at LADIS (Rodrigues et al. [30], [31]) presented a specification of exactly such a service. Such steps could be fertile areas for further study: topics close enough to today's hot areas to publish upon, and yet directly relevant to cloud computing.

3. Not enough is known about stability of large-scale event notification platforms, management technologies, or other cloud computing solutions. As we scale these kinds of tools to encompass hundreds or thousands of nodes spread over perhaps tens of data centers, worldwide, we as researchers can't help but be puzzled: how do our solutions work today, in such settings?
    - Very large-scale eventing deployments are known to be prone to destabilizing behavior – a communications-level equivalent of thrashing. Not known are the conditions that trigger such thrashing, the best ways to avoid it, the general styles of protocols that might be inherently robust or inherently fragile, etc.
    - Not very much is known about testing protocols to determine their scalability. If we invent a solution, how can we demonstrate its relevance without first taking leave of our day jobs and signing on at Amazon, Google, MSN or Yahoo? Today, realistically, it seems nearly impossible to validate scalable protocols without working at some company that operates a massive but proprietary infrastructure.
    - Another emerging research direction looks into studying subscription patterns exhibited by the nodes participating in a large-scale publish-subscribe system. Researchers (including the authors of this article) are finding that in real-world workloads, the subscription patters associated with individual nodes are highly correlated, forming clusters of nearly identical or highly similar subscriptions. These structures can be discovered and exploited (through e.g., overlay network clustering [9], [17], or channelization [35]). LADIS researchers reported on opportunities to amortize message dissemination costs by aggregating multiple topics and nodes, with the potential of dramatically improving scalability and stability of a pub-sub system.
4. Our third point leads to an idea that Mahesh Balakrishnan has promoted: we should perhaps begin to treat virtualization as a first-class research topic even with respect to seemingly remote questions such as the scalability of an eventing solution or a tolerating Byzantine failures. The point Mahesh makes runs roughly as follows:
    - For reasons of cost management and platform management, the data center of the future seems likely to be fully virtualized.
    - Today, one assumes that it makes no sense to talk about a scalable protocol that was actually evaluated on 200 virtual nodes hosted on 4 physical ones: one presumes that internal scheduling and contention effects could be more dominant than the scalability of the protocols per se. But perhaps tomorrow, it will make no sense to talk about protocols that *aren't* designed for virtualized settings in which nodes will often be co-located. After all, if Hamilton is right and cost factors will dominate all other decisions in all situations, how could this not be true for nodes too?

    - Are there deep architectural principles waiting to be uncovered – perhaps even entirely new operating systems or virtualization architectures – when one thinks about support for massively scalable protocols running in such settings?

5. In contrast to enterprise systems, the only economically sustainable way of supporting Internet scale services is to employ a huge hardware base consisting entirely of cheap off-the-shelf hardware components, such as low-end PC's and network switches. As Hamilton pointed out, this reflects simple *economies of scale*: i.e., it is much cheaper to obtain the necessary computational and storage power by putting together a bunch of inexpensive PC's than to invest into a high-end enterprise level equipment, such as a mainframe. This trend has important architectural implications for cloud platform design:
    - Scalability emerges as a crosscutting concern affecting all the building blocks used in cloud settings (and not restricted to those requiring strong consistency). Those blocks should be either redesigned with scalability in mind (e.g., by using peer-to-peer techniques and/or dynamically adjustable partitioning), or replaced with new middleware abstractions known to perform well when scaled out.
    - As we scale systems up, sheer numbers confront us with growing frequency of faults within the cloud platform as a whole. Consequently, cloud services must be designed under assumption that they will experience frequent and often unpredictable failures. Services must recover from failures autonomously (without human intervention), and this implies that cloud computing platforms must offer standard, simple and fast recovery procedures [18]. We pointed to a seeming connection to recovery oriented computing (ROC) [27], yet ROC was proposed in much smaller scale settings. A rigorously specified, scalable form of ROC is very much needed.
    - Those of us who design protocols for cloud settings may need to think hard about churn and handling of other forms of sudden disruptions, such as sudden load surges. Existing protocols are too often prone to destabilized behaviors such as oscillation, and this may prevent their use in large data centers, where such events run the risk of disrupting even applications that don't use those protocols directly.

We could go on at some length, but these points already touch on the highlights we gleaned from the LADIS workshop. Clearly, cloud computing is here to stay, and poses tremendously interesting research questions and opportunities. The distributed systems community, up until now at least, owns just a portion of this research space (indeed, some of the topics mentioned above are entirely outside of our area, or at best tangential).

LADIS 2009

In conclusion, LADIS 2008 seems to have been an unqualified success, and indeed, a far more thought-provoking workshop than we three have attended in some time. The key was that LADIS generated spirited dialog between distributed systems researchers and practitioners, but also that the particular practitioners who participated shared so much of our background and experience. When researchers and system builders meet, there is often an impedance mismatch, but in the case of LADIS 2008 we managed to fill a room with people who share a common background and way of thinking, and yet see the cloud computing challenge from very distinct perspectives.

LADIS 2009 is now being planned running just before the ACM Symposium on Operating Systems in October 2009, at Big Sky Resort in Utah. In contrast to the two previous workshops, the papers are being solicited through both an open Call For Papers, and targeted solicitation. If SOSP 2009 isn't already enough of an attraction, we would hope that readers of this essay might consider LADIS 2009 to be absolutely irresistible! You are most cordially invited to submit a paper and attend the workshop. More information can be found at http://www.sigops.org/sosp/sosp09/workshops-cfp/ladis09-cfp.pdf and http://www.cs.cornell.edu/projects/ladis2009.

References

[1] Aguilera, M. K., Merchant, A., Shah, M., Veitch, A., & Karamanolis, C. (2007). Sinfonia: a new paradigm for building scalable distributed systems. *SOSP '07: Proceedings of twenty-first ACM SIGOPS Symposium on Operating Systems Principles* (pp. 159-174). Stevenson, Washington: ACM.

[2] Amazon.com. *Amazon Simple Storage Service (Amazon S3)*. http://aws.amazon.com/s3/

[3] Apache.org. *HDFS Architecture*. http://hadoop.apache.org/core/docs/current/hdfs_design.html

[4] Burrows, M. (2006). The Chubby lock service for loosely-coupled distributed systems. *OSDI '06: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation* (pp. 24-24). Seattle, WA: USENIX Association.

[5] Candea, G., & Fox, A. (2003). Crash-only software. *HOTOS'03: Proceedings of the 9th conference on Hot Topics in Operating Systems* (pp. 12-12). Lihue, Hawaii: USENIX Association.

[6] Castro, M., & Liskov, B. (1999). Practical Byzantine fault tolerance. *OSDI '99: Proceedings of the third Symposium on Operating Systems Design and Implementation* (pp. 173-186). New Orleans, Louisiana: USENIX Association.

[7] Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., & Gruber, R.E. Bigtable: A Distributed Storage System for Structured Data. OSDI'06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, November, 2006.

[8] Chockler, G., Keidar, I., & Vitenberg, R. (2001). Group communication specifications: a comprehensive study. *ACM Computing Surveys, 33* (4), 427-469.

[9] Chockler, G., Melamed, R., Tock, Y., & Vitenberg, R. SpiderCast: a scalable interest-aware overlay for topic-based pub/sub communication. *DEBS '07: Proceedings of the 2007 inaugural International Conference on Distributed Event-Based Systems.* Toronto, Ontario, Canada: ACM, 2007. 14-25.

[10] Clement, A., Marchetti, M., Wong, E., Alvisi, L., & Dahlin, M. (2008). BFT: the Time is Now. *Second Workshop on Large-Scale Distributed Systems and Middleware (LADIS 2008).* Yorktown Heights, NY: ACM. ISBN: 978-1-60558-296-2**.**

[11] Dean, J., & Ghemawat, S. (2004). MapReduce: simplified data processing on large clusters. *OSDI'04: Proceedings of the 6th Symposium on Operating Systems Design and Implementation* (pp. 10-10). San Francisco, CA: USENIX Association.

[12] Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Commun. ACM , 51* (1), 107-113.

[13] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., et al. (2007). Dynamo: Amazon's highly available key-value store. *SOSP '07: Proceedings of the twenty-first ACM SIGOPS Symposium on Operating Systems Principles* (pp. 205-220). Stevenson, Washington: ACM.

[14] Devlin, B., Gray, J., Laing, B., & Spix, G. (1999). *Scalability Terminology: Farms, Clones, Partitions, and Packs: RACS and RAPS* . Microsoft Research Technical Report MS-TR-99-85. Available from ftp://ftp.research.microsoft.com/pub/tr/tr-99-85.doc.

[15] Fan, X., Weber, W.-D., & Barroso, L.A.. Power provisioning for a warehouse-sized computer. *ISCA '07: Proceedings of the 34th annual International Symposium on Computer Architecture.* San Diego, California, USA: ACM, 2007. Pp. 13-23.

[16] Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). The Google File System. *SOSP '03: Proceedings of the nineteenth ACM Symposium on Operating Systems Principles* (pp. 29-43). Bolton Landing, NY: ACM.

[17] Girdzijauskas, S., Chockler, G., Melamed, R., & Tock, Y. Gravity: An Interest-Aware Publish/Subscribe System Based on Structured Overlays (fast abstract). *DEBS 2008, The 2nd International Conference on Distributed Event-Based Systems.* Rome, Italy.

[18] Hamilton, J. Perspectives: James Hamilton's blog at http://perspectives.mvdirona.com/

[19] Hamilton, J. (2007). On designing and deploying Internet-scale services. *LISA'07: Proceedings of the 21st conference on Large Installation System Administration Conference* (pp. 1-12). Dallas, TX: USENIX Association.

[20] Kirsch, J., & Amir, Y. (2008). Paxos for System Builders: An Overview. *Second Workshop on Large-Scale Distributed Systems and Middleware (LADIS 2008).* Yorktown Heights, NY: ACM. ISBN: 978-1-60558-296-2.

[21] Kotla, R., Alvisi, L., Dahlin, M., Clement, A., & Wong, E. (2007). Zyzzyva: speculative Byzantine fault tolerance. *SOSP '07: Proceedings of twenty-first ACM SIGOPS Symposium on Operating Systems Principles* (pp. 45-58). Stevenson, WA: ACM.

[22] *LADIS 2008: Proceedings of the Second Workshop on Large-Scale Distributed Systems and Middleware.* (2009). Yorktown Heights, NY, USA: ACM International Conference Proceedings Series. ISBN: 978-1-60558-296-2.

[23] *LADIS 2008: Presentations and Related Material.* http://www.cs.cornell.edu/projects/ladis2008/presentations.htm

[24] Lamport, L. (1998). The part-time parliament. *ACM Trans. Comput. Syst. , 16* (2), 133-169.

[25] MacCormick, J., Murphy, N., Najork, M., Thekkath, C. A., & Zhou, L. (2004). Boxwood: abstractions as the foundation for storage infrastructure. *OSDI'04: Proceedings of the 6th*

conference on Symposium on Operating Systems Design & Implementation (pp. 8-8). San Francisco, CA: USENIX Association.

[26] *memcached: a Distributed Memory Object Caching System*. http://www.danga.com/memcached/

[27] Patterson, D. *Recovery Oriented Computing*. Retrieved from http://roc.cs.berkeley.edu

[28] Reed, B., & Junqueira, F. P. (2008). A simple totally ordered broadcast protocol. *Second Workshop on Large-Scale Distributed Systems and Middleware (LADIS 2008).* Yorktown Heights, NY: ACM. ISBN: 978-1-60558-296-2.

[29] Shoup, Randy. (2007) Randy Shoup on eBay's Architectural Principles. San Francisco, CA, USA. http://www.infoq.com/presentations/shoup-ebay-architectural-principles. A related video is available at http://www.se-radio.net/podcast/2008-09/episode-109-ebay039s-architecture-principles-randy-shoup.

[30] Singh, A., Fonseca, P., Kuznetsov, P., Rodrigues, R., & Maniatis, P. (2008). Defining Weakly Consistent Byzantine Fault-Tolerant Services. *Second Workshop on Large-Scale Distributed Systems and Middleware (LADIS 2008).* Yorktown Heights, NY: ACM. ISBN: 978-1-60558-296-2.

[31] Singh, A., Fonseca, P., Kuznetsov, P., Rodrigues, R., & Maniatis, P. (2009). Zeno: Eventually Consistent Byzantine Fault Tolerance. *Proceedings of USENIX Networked Systems Design and Implementation (NSDI).* Boston, MA: USENIX Association.

[32] Terry, D. B., Theimer, M. M., Petersen, K., Demers, A. J., Spreitzer, M. J., & Hauser, C. H. (1995). Managing update conflicts in Bayou, a weakly connected replicated storage system. *SOSP '95: Proceedings of the fifteenth ACM Symposium on Operating Systems Principles* (pp. 172-182). Copper Mountain, Colorado: ACM.

[33] Van Renesse, R., Birman, K. P., & Vogels, W. (2003). Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst. , 21* (2), 164-206.

[34] Van Renesse, R., Dumitriu, D., Gough, V., & Thomas, C. (2008). Efficient Reconciliation and Flow Control for Anti-Entropy Protocols. *Second Workshop on Large-Scale Distributed Systems and Middleware (LADIS 2008).* Yorktown Heights, NY: ACM. ISBN: 978-1-60558-296-2.

[35] Vigfusson, Y., Abu-Libdeh, H., Balakrishnan, M., Birman, K., & Tock, Y. Dr. Multicast: Rx for Datacenter Communication Scalability. *HotNets VII: Seventh ACM Workshop on Hot Topics in Networks.* ACM, 2008.

[36] Yalagandula, P. and Dahlin, M. A Scalable Distributed Information Management System. ACM SIGCOMM, August, 2004. Portland, Oregon.

[37] Yu, Y., Isard, M., Fetterly, D., Budiu, M., Erlingsson, U., Gunda, P. K., et al. (2008). DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language. *Symposium on Operating System Design and Implementation (OSDI).* San Diego, CA, December 8-10, 2008. http://research.microsoft.com/en-us/projects/DryadLINQ.