Final

CMSC 818e: Distributed and Cloud-Based Storage Systems

Due 5pm Wednesday, December 16, 2015

Name Josh Reese (107768421)

Instructions

- You may have **one free peek** at the exam for 10 minutes at some point before you start taking it. The peek does not even have to be the same day as you take it.
- The exam is **open everything**.
- Answers should average 3/4 to a full page each.
- You have 120 minutes to complete the exam once you start taking it.
- The exam is worth **100** points.
- Write neatly. Credit cannot be given for illegible answers.
- Please **email me your exam solution as a PDF.** Scan it in, take (good) pictures, or latex your answer, for example. The only important factor is that you send me an email by 5pm Wednesday.

1. (20 pts) Spanner and PBS (Bailis) seem to be heading in different directions. How will this play out out over the next five years?

Spanner is Google's multi-version, synchronously replicated, externally consistent distributed semi-relational database. It provides non-blocking reads and read-only transactions with the ability to scale to millions of machines across the globe. Much of this is facilitated by their TrueTime API. On the other hand PBS is an evaluation of how weak-consistency is good enough and how various specifics of a weakly-consistent system can be tuned to provide the desired balance between consistency and latency. Their claim is that weak-consistency general provides the correct answer, so why slow things down trying to get better consistency guarantees?

One question that needs to asked when looking at the results PBS presents is, how will these results change as the scale of distribution grows? Once it becomes common place to have data centers located all around the world with millions of machines is it the case that weak consistency will still provide the same results? This seems unlikely. If a user is operating in one country while replication needs to be distributed far around the globe the chances of getting the most recent version of data seems as though it will get less and less. In addition to this, if a system is using millions of machines replication is going to take much longer which means updates will be visible much less frequently. In the future, as the scale of systems continue to grow, it seems unlikely that weak-consistency will continue to provide the same positive results that it provides currently.

If strong consistency is realizable with reasonable amounts of latency, why shouldn't it be provided? The TrueTime API provided by Spanner is absolutely revolutionary. A constraint that distributed systems have been working around for a long time is now, to some extent, destroyed by this functionality. By synchronizing a system based on the real-time happenings of events Spanner is able to provide much stronger guarantees without much loss to latency. As systems continue to grow, both globally and internally, the idea of being able to look at the *actual* time of transactions will greatly increase the reliability of the system. In addition to this Spanner gives the user control over how they want replication to be facilitated. If latency becomes a bottleneck in the system the user can configure which datacenter will hold replicas, how far the replicas are locating geographically from one another and the amount of replicas to maintain. Given the ability to control these parameters Spanner is able to provide guarantees on reliability but give the user the ability to control the latency, durability and availability of their system.

In the long run eventual-consistency is no longer going to be "good enough". Systems will grow to be too large (and inherent latencies forced by actual distance between machines and amount of machines being used) for weak-consistency to be able to provide the necessary functionality. As it appears Google is providing a system with much stronger guarantees and the ability to control enough of it to tune the system to fit a user's needs. If we can get stronger consistency, more reliability and durability, shouldn't we do that?

2. (20 pts) Overloading file system namespaces a la “Semantic File Systems” seems to be a way to roll out new system functionality to all applications, including legacy. What functionality could you see being useful?

Semantic filesystems provide content based file structuring, as opposed to name based structuring, in the design of a filesystem. These systems provide fast lookups of files (due to automatic indexing), they are transparent to users and integrate with legacy filesystems. In addition they make locating files much more robust as users can specify meta information about files to gather results. This could be something like, “give me all the files owned by Josh” or “give me all the files which are owned by Josh and are pictures of dogs”, etc. These systems also provide this content based access on a distributed scale.

One interesting feature that these systems provide is the automatic extraction of indexes based on content filters. In the Semantic FS literature these are referred to as transducers, which are filters that take the contents of a file and output the corresponding attributes. Given these transducers the filesystem is able to create indexes which allow querying based on content, as the indexes contain information about the content, based on these transducers. While this is interesting enough, what is really useful here is that these transducers can be programmed by the user allowing for creation of new filters and thus expanding the functionality of the system.

With the modification introduced by transducers and querying based on content looking up files becomes very different. Since the indexes are built automatically based on the transducers, performing a content based query becomes much faster since the system only needs to scan the indexes. Since querying a filesystem happens pretty frequently this is certainly a useful feature. On top of this one can argue that queries can become more intuitive and robust. The user will be asking the filesystem for results based on the contents of what they are actually looking for as opposed to arbitrary names of items. Once again, this makes searching the filesystem easier and more flexible.

They key advantageous functionality provided by this system is it’s transparency. No one is going to want to switch entirely to a new filesystem when we have been using namespace filesystems all our lives. The overhead to learn how a new system works, the fear of unreliability and the migration of data from one system to the next would absolutely kill this immediately. However, they have provided functionality to make the semantic filesystem transparent to the user with their additional features presumably built on top as a bonus. Not only this, but they support legacy filesystems so users won’t need to worry about trying to deal with an impedance mismatch between whatever system they were using before and this new system.

Again, the key advantage to something like this is that it provide *new* features without the loss of *old* features. Additionally, users are able to use their current platform in conjunction with this new platform. I’m not sure if any of the papers talked about this, but I wonder what security features could be leveraged to use this approach

3. (20 pts) Perspective and Cimbiosys seem almost to be one-offs: they explicitly limit the scope of their systems to be personal storage. What would it take to extend these systems to enable explicit collaboration among users?

Perspective is an “at home” storage system design for a non-technical user. This allows users to be their own administrators without the overhead of having to know too much about how the system actually works. They provide “views” which are a semantic description of how files are stored on various devices. Cimbiosys is also designed for home media management with content based filtering for various devices.

One clear disadvantage to these systems is there appears to be no support for multiple users. Perhaps this isn’t a concern for what they are trying to accomplish but if users are to collaborate, this is a must. We can imagine a scenario where two users are accessing the same file and attempting to make modifications. These modifications could be to the content itself or to the metadata associated with it. In addition, users might want to change the content filters on various devices while other users are using the system as well. These sorts of operations could potentially cause outdated files to be viewed, or files to not be able to be located properly, or even storage of items to be lost or stored on incorrect devices.

If these systems are to support these types of operations they will need some concept of consistency between the various devices. This could come in the obvious form of locking files if they are being modified and notifying devices if their content becomes stale. Perhaps there could be some notion of transactions. This would likely need to be very lightweight because it’s probably not the case that there are a lot of modifications happening concurrently in the system. However, if modifications do happen concurrently the system should be able to handle them.

Another clear limitation to these systems, as discussed in class, is the lack of description of how these might actually be used. If someone wanted to develop an app to utilize this type of storage how would they do that? What is provided by the system and what do the users need to do to retrieve and store files? If consistency was needed would this be implemented in the filesystem or is this up to the users to deal with? The whole notion of usability is very nebulous in this work, which makes collaborating in this environment very difficult.

The idea behind these systems is appealing. It would certainly be nice to have a way to easily share information between devices and have a easy way to manage what is stored where but they are lacking in a number of ways.

4. (20 pts) The P2P approach seems to have been trumped by the cloud revolution, at least in the business world. Why is this, and where will we be in five years?

In order for a P2P system to be feasibly used in a critical distributed environment it must provide availability, durability, access control and authenticity. While there are arguments to be made that P2P systems can provide these it is perhaps harder to envision this happening in a P2P system as opposed to a controlled cloud environment.

One key disadvantage to a P2P system is that the guarantees provided are purely probabilistic. In an environment where machines can join and leave whenever they want it is virtually impossible to provide any static guarantees about how the system will perform. For many applications this is totally unacceptable. The entire design of Amazon's Dynamo is the complete opposite of this and is precisely what they need. They need very controlled SLAs and have designed their system around this, while it's unlikely they can get this from a P2P system. There are many other systems that require this type of predictable guarantee on performance. These guarantees aren't only intrinsic to performance but to security as well. If a cloud provider owns all of it's machines and can trust them, they can provide a level of security that would be hard, if not impossible, to provide in a P2P setting. Every machine that joins a P2P system is, by nature, untrusted while every machine controlled by the cloud provider can be trusted.

This doesn't say anything about availability. In a P2P system there are a number of challenges that revolve around making sure data is always available. How is the data stored and replicated? What if a machine storing some data leaves? Does that data get migrated to an existing machine and what if modifications are being made to it while this is happening? The loose nature of the P2P environment makes replication and availability somewhat tricky because machines are able to join and leave at will, and this is the desired functionality of this type of environment. Back to security again, even if data is available all the time it's unclear whether or not it can be trusted. Any machine is able to corrupt it's data since it's untrusted so it could certainly flood the system with inaccurate data.

In the upcoming years it seems that this push towards cloud centric systems will continue to grow. Revisiting the idea of cloud providers owning their entire system is a clear indication of this. If Google or Amazon has total control over their entire system they are able to concretely provide guarantees that are impossible to provide in a P2P system. As systems continue to grow and security continues to be of paramount importance it's unclear whether systems designed to work in a P2P setting will be able to be realizable. However, there are certainly applications which could work very well in a P2P distributed environment, and some are already out there. For example, the BOINC project works by distributed workloads to machines in a P2P setting. This is fine because guarantees on latency or reliability aren't that important. As long as the get a result and can verify, by comparing to other results, the accuracy this is enough. One could imagine other scientific computing settings where this might also be desirable.

5. (20 pts) Say you were tasked by your employer to build a free Dropbox Killer. Describe the system you would build.

Do you feel limited by Dropbox? Do you want more from your cloud storage system? If so, download JoshBox now! We provide all of the features present in the current Dropbox system (cloud storage, easy access, easy sharing, file encryption and mobile access) but with so much more!

One key disadvantage of Dropbox is the inability to work collaboratively with a group. In JoshBox we offer the ability to edit files in real-time with other users. Open a file, begin editing and view the changes from any other device able to access this file. Not only that, but you can edit the same file with other users connected and editing at the same time. Unlike GoogleDrive this doesn't need to be done from some messy browser based editor but can be edited by and application present on your machine. This allows for concurrent editing of any type of file.

If your not sold yet, what if you could get more storage for less? In the current version of Dropbox a user is allowed 2GB of storage. If they want more they have to upgrade their membership which allows them 100GB of storage. Many users need a bit more than 2GB but way less than 100GB. In JoshBox we offer a 'pay what you use' membership. Just like in Dropbox each user is given 2GB of free storage. However, by utilizing Amazon's S3 service we provide extremely inexpensive storage for any amount of data.

For our more advanced users we provide two new features: fine-grained access control and fine-grained replication. Let's imagine a scenario where a user is working on in a shared directory with several other users but wants to make a file only accessible by a subset of those users. While this is currently not supported in many cloud based storage systems, it is indeed provided by JoshBox. No longer are you restricted by the inability to share exactly what you want to share.

Our most advanced, and novel feature, is fine-grained replication. Some of your devices may have less storage than others and it is imperative that your entire shared directory not be copied onto them. In JoshBox we offer the user the ability to specify, from a list of registered machines, files that shouldn't be synced to specific machines. This gives users the flexibility to determine exactly where their files will be accessible. Finally, do you want to have the flexibility to create your own cloud? Are you too paranoid to put all your data into the 'cloud'? JoshBox provides a feature which allows the user to set up their own server as a backup for their files. This allows users to build a cloud of their own, totally bypassing the storage provided by JoshBox and eliminating any storage limits, while maintaining total privacy.

We hope you will check out JoshBox and look forward to hearing from you.