# Extreme Computing - First Practical Assignment

s1006260

November 11, 2013

## 1 Introduction

This essay shall employ a top-down approach to attempt to design an infrastructure appropriate for use by Massbase. This shall be achieved by identifying the use-cases of the Massbase system, considering existing approaches for dealing with these use-cases, and evaluating them with respect to the overall system. The essay shall conclude by proposing a full infrastructure for the Massbase system, drawing the best solutions to the various use-cases together into one full infrastructure.

## 2 Design Considerations

To begin the use-cases of the Massbase system must be identified. It may not be realistic to try and satisfy all use-cases; some may be contradictory in nature, or may not be practically possible with current technology. The use-cases for the Massbase system are:

- Monitor social networking sites for blips
- Tag blips with user and location
- Store the blips
- Perform data de-duplication on blips

  - This can be done either in the real time or later on

- Tags should be stored separately from blips, but should not lose their connection to the blip
- Some post-processing should be done on blips to identify additional tags
  - This need not be instant, but should be take no more than two hours
  - This can be done in bulk
- Networks of users should be identified by looking at tags
- Aggregate across multiple dimensions
  - These computations should be performed in bulk
  - Results should be stored efficiently for fast retrieval
- Trends should be identified by looking at tags
  - Tag disambiguation should occur
- Users should be sent messages to alert them to related information
  - This can be done real time, or in bulk (i.e. overnight)
- Recently used tags should have higher importance than older tags
- The system should be always running
- The system should be elastic

From this we can make several key observations. Firstly, any files stored are likely to be small; Tweets have a 140 character limit, and Facebook/Google+ statuses, while longer, tend not to be large. Additionally, tags tend to be just one word long. This means that any records stored in the Massbase system will be small, if not tiny.

Secondly, while these records may well be very small, there are likely to be a very large number of them. Massbase will want to service as many users as possible, and, looking from the other direction, users of the Massbase system are likely to make many contributions to various social media sites. Together, these two facts mean that there will be a substantially large number of blips, meaning a large number of records.

Thirdly, these records are, at best, semi-structured. Whilst we know that each blip is related to it's tags, we do not know in advance how many tags there will be, as this is a result of post-processing on the blip.

We also know that the system will likely perform more read operations than write operations. Blips and their corresponding tags need be written only once, however may be read many times when they are processed for the purposes of identifying trends, tag disambiguation and identifying networks of users.

Additionally, many of the tasks performed by the system lend themselves very well to being processed in bulk, and need not be performed in real time.

Furthermore, it is also worth remembering that Massbase is a startup company.

# 3 Infrastructure Options

With this list of use cases, and these observations, we can begin to design our infrastructure.

## 3.1 Database or Cloud Infrastructure?

The first thing we must consider is whether to use a database infrastructure or a cloud based infrastructure. Whilst a solution could probably be built using either approach, the problem at hand seems to lend itself very well to a cloud infrastructure, with little to recommend a database infrastructure.

The main reason for this is the fact that we are operating on semi-structured data. While techniques exist for handling semi-structured data using a database, they are mostly somewhat awkward workarounds involving the use of null values. Cloud based approaches, such as Hadoop Map-reduce, can operate on semi-structured data much more naturally.

Another reason for wanting to adopt a cloud based approach is that Map-Reduce algorithms are perfect for bulk processing jobs, which is what Massbase is primarily doing. In their article [1] Stonebraker et al. state that "[they] expect ETL and complex analytics to be amenable to MR systems and query-intensive workloads to be run by DBMSs". The Massbase system fits the profile of ETL (Extract, Transform, Load) and complex analytics

much more that it fits the profile of being query-intensive, meaning an MR (map reduce) approach is probably more suitable.

In the same article [1] Stonebraker et al. also state that "The out-of-the-box experience for most DBMSs is less than ideal" and that "commercial DBMS must move more towards one button installs". Given that Massbasse is a new startup company they might prefer to have an infrastructure up and running quickly rather than having to take more time setting up a more complicated DBMS system.

A hybrid approach might be possible, but would probably not offer anything much beyond what a purely cloud based infrastructure could offer. Indeed, even after post-processing to identify implicit, data would still be semi-structured as different numbers of implicit tags could be identified for each blip.

## 3.2 To Buy or To Rent?

Given that a cloud based approach seems more suitable, the next decision is whether it would be more beneficial to Massbase to set up their own private cloud or to rent the resources from a vendor. Both approaches allow for some amount of elasticity with regards to dynamic resource allocation, however the potential for this is greater with the renting approach. Setting up their own private cloud would give Massbase a static amount of resource to allocate to tasks, and while this static amount of resource could be allocated dynamically, the amount of resource itself cannot change. There is potential that this could lead to resources being idle if the infrastructure is too big for the job at hand, or it could lead to resources being competed for by different jobs, leading to some jobs being neglected.

For example, if the website grows faster than anticipated and the infrastructure can't keep up, this could lead to bulk jobs such as trend detection, tag disambiguation and user network detection being neglected, resulting in a slower service for users.

Alternatively, the opposite could occur and resources could be wasted. Given that Massbase is a new startup company, usage of it's service will start low, and will grow as word spreads about it. This will mean that, initially, resource demand will likely be low. However, the infrastructure will have been designed with a much higher demand in mind, meaning that for a period of time resources will likely be wasted, which is not very cost efficient. It could be made more viable if Massbase had some kind of guarantee on

user numbers. This could perhaps be somewhat achieved; Beta testing would allow people to get to know the product and gauge interest levels, and good (possibly quite aggressive) marketing could help ensure that a good number of early adopters of the system. However it still seems a somewhat reckless approach.

Renting from a vendor, on the other hand, would allow Massbase to start small and then scale up their system to match demand. This would help to avoid high setup costs and wasted resource. Renting from a vendor also means means that rather than neglecting bulk jobs during busy periods, they could simply rent more resource. This is a higher level of elasticity than is offered by having a private cloud.

Of course a private cloud can be extended - more nodes can be added. However this is a lot more difficult than simply renting more from a vendor. The process of adding more nodes, or even more clusters, would certainly be slower, as it would require physical engineers to set them up and physical space would have to be acquired (either bought or leased) for the cluster/nodes.

Additionally, this process could not really be automated, a disadvantage as automation in this regard is fast becoming possible. In fact,tools already exist to automate control of elastic resource provisioning for cloud applications - Amazon, for example, already offer an autoscaling service [2], with Google and Microsoft offer similar functionality [3]. Other tools are also being developed. One such example is the Celar project, and open source project aiming to "deliver a fully automated and highly customisable system for elastic provisioning of resources in cloud computing platforms" [4]. Renting from a vendor, then, has the potential for automating elastic resource provisioning, where building your own private cloud does not.

This automated elastic provision of resources helps to ensure that the service can be constantly running and available. If Massbase were to experience very heavy traffic, to the point where it might go down, more resources would automatically be allocated. This has to be considered quite carefully, however, as it could make the service susceptible to a bankrupting attack [5]. It may be that this requirement has to be revised slightly.

Given the requirement for elasticity, then, and the additional capabilities of renting with regards to this, it would likely be best for the company to rent resources from a vendor. This would also avoid the problem of having high start up costs.

## 3.3 File System

Underlying any cloud based architecture, there must be some kind of distributed file system. There are many different ones to choose from, each with their own advantages and disadvantages, some catering to very specific needs. We observed earlier that the Massbase system will be working on a large number of small files, and that reads would likely be more common than writes. These observations would seem to suggest that the Andrew File System (AFS) might be a good fit, as it was designed for working with large numbers of small files for which reads are more common than writes.

AFS could provide an easy way of fulfilling the requirement that blips should be stored independently of tags. Each user could have a directory, and within that directory new directory could be made for each blip. Within the blip directory could be separate files for the blip itself, and the tags associated with the blip. This would create a tree like structure of directories.

AFS also has great scalability - this is one of the reasons that it is the most commonly deployed distributed filesystem. This means that Massbase could grow to virtually any size and AFS would still serve as a file system.

In terms of fault tolerance, AFS records can be replicated to read only copies to provide backups should the main copy fail. This means that, with a sufficient amount of replication, AFS can be fault tolerant enough for there to be an acceptably small probability of data loss.

AFS also has several nice features relating to security, however these do not seem particularly relevant to the Massbase system.

There is certainly a strong case to be made for using AFS as the file system for Massbase. However, if Massbase is renting resources from a vendor, it would be nice to take full advantage of the services offered by that vendors. Most vendors use a variant of HDFS, and have optimised their services to such an extent that they run faster than a standard implementation of Hadoop and Map-Reduce would.

HDFS, (or GFS, which HDFS is based off of), was built with very different design goals in mind. It assumes large file sizes, with files being stored on chunk servers. Each cluster has several chunk servers, served by one master. This framework means the master is a single point of failure, however they are normally replicated to a read only shadow master, so that the system can quickly recover in the event of a failure. The master can be a bottleneck device, to alleviate this only metadata is passed through the master.

While this might seem unsuitable as we are dealing with small files, rather than write them to the filesystem individually, HDFS uses append operations instead. This means that the many small files are concatenated into substantially fewer, much larger files.

This does have the unfortunate consequence that we are not storing tags independently of of blips any longer; indeed, even blips themselves are not distinct. However as a system it will still be functional.

Additionally, due to the fact that HDFS can only really read files in a forward direction, this has the effect of making recently used tags more important than older ones - as we can always skip through the the file, but we can't read it backwards. Worded another way, if we read the old tags, we will also read the newer tags, whereas the converse is not true.

Using HDFS as a filesystem is also a viable option, then, and while it might be a little more awkward from the perspective of Massbase, it will most likely integrate more easily with the vendor, and possibly improve performance.

## 3.4 Data De-duplication

Data de-duplication could be achieved in one of two ways [6]. It could either be done in post processing (removing duplicates after they are stored), or it could be done in-line (checking to see whether the data is a duplicate before storing it). In both cases, this is checked using hash functions. If 2 records hash to the same value, they are likely to be the same. Some systems check the contents of the records before discarding and some do not.

In-line data de-duplication minimises disk operations, as duplicates or not stored and then subsequently deleted, however it also slows down the overall rate of data intake. Conversely, post-processing can have a higher rate of data intake, but requires many more disk operations.

I propose that Massbase use an In-Line data de-duplication strategy. While either strategy is viable, and both have their advantages/disadvantages, with there already being so many batch jobs being performed on the disk I think that if we can eliminate one, even if it slows down data intake, then we should.

This may turn out to be the wrong suggestion. If data intake proved to be too slow, Massbase could switch to a post-processing de-duplication strategy relatively simply. It may be wise to try both and see which one performs better.

### 3.5 User Recommendations

In the brief Massbase envisaged sending recommendations to users as to things they may be interested based upon their blips. They were not sure whether to do this in real-time or in bulk.

I would propose that they do this in bulk. I do not think that doing it in real-time, at least with the infrastructure I have provided, is realistic; blips require post-processing to generate tags from which many of these recommendations might be drawn, and as such any real-time recommendations may not be very good. It may be possible to generate *some* recommendation in real-time, using the explicit tags, however not as much information can be drawn from these. I think the system would be much more effective if it sent the users emails with recommendations overnight.

The other problem with real-time recommendations is that, taking into account the frequency with which some people use social networking services, (the frequency with which they generate blips), the repeated recommendations would likely get very irritating. A bulk approach to this is much safer.

## 4 Proposed Solution

My proposed solution, then, is to use a cloud based infrastructure, making use of the Map Reduce algorithm as it lends itself quite well the the types of processing we are performing. I propose that Massbase should rent resources from an existing vendor, rather than try and build up it's own cloud infrastructure, as this would allow them to automate elastic resource provisioning (if they chose a vendor which offered this service), which is a very powerful idea. For a file system I propose that Massbase use the Andrew File System. While there are advantages to both AFS and HDFS, and both would work quite adequately, I feel that AFS fits the data more naturally than HDFS, which is a little less natural. This may mean it is a little more difficult to integrate with the vendor, however, and if problems occur then HDFS could be used as a strong alternative. I propose that Massbase use some form of hash function to perform in-line data de-duplication, however I recommend that they monitor the performance of this solution, and, if it proves inadequate, switch to a post-processing method of data de-duplication. Finally, I propose that the recommender system be done in bulk rather than real time, as this fits the proposed infrastructure better and almost certainly provide better and more useful recommendations for the user.

# 5 Conclusion

In conclusion I think I have presented a reasonable plan for Massbase to design an infrastructure. I have tailored the infrastructure to the system as best I can, satisfying as many requirements as I reasonably could. I have identified some alternative solutions to some use-cases, so that Massbase has some idea of what they might wish to change if the system does not work to a high enough standard.

# References


[1] Stonebraker, M; Abadi, D; Dewitt, D; Madden, S; Paulson, E; Pavlo, A; Rasin, A. (2010) MapReduce and Parallel DBMSs: Friends or Foes? *Communications Of The Acm.* Volume 53 (1), Pages 64-71.

[2] Auto Scaling, Developer Guide (API Version 2011-01-01). http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/WhatIsAutoScaling.html

[3] Butler, B. (2013) Google, Microsoft play catch up to Amazon, add load balancing, auto-scaling to their clouds, http://www.networkworld.com/news/2013/080713-google-microsoft-cloud-272620.html .

[4] Automatic, Multi-Grained Elasticity-Provisioning for the Cloud, http://www.celarcloud.eu/.

[5] Salam, A. (2013) Auto Scaling In Cloud Computing, http://www.cloudtweaks.com/2013/06/auto-scaling-in-cloud-computing/ .

[6] Curtis, W. (2007) In-line or post-process de-duplication?, http://www.backupcentral.com/mr-backup-blog-mainmenu-47/13-mr-backup-blog/134-inline-or-post-process.html .