Extreme Computing

First Practical Assignment

Essay on the infrastructure and uses of cloud computing

Name: Georgi Tsatsev

Matriculation Number: S1045049

INTRODUCTION

In this essay I have provided an infrastructural design for the large-scale recommendation engine according to the set of specifications provided by the Massbase Company. My choice for the proposed system is based on varying sources including information on parallel databases, cloud based infrastructures, information on existing cloud providers and the economics and obstacles for renting supercomputers for the task at hand. Other part of the essay is relating to the Hadoop framework for storing and processing large scale data-sets. The aim of this proposal is to provide a distributed system that takes into consideration most of the issues that come up with the specifications and tackle them accordingly. These issues are connectivity, replication, fault tolerance, file access, capabilities etc.

More specifically I have considered renting an existing cloud as opposed to building and owning a private one, based on the assumption that it is hard to predict how much resources would be needed for such system and it more scalable this way. Advantages and disadvantages of the structured parallel database and the cloud-based infrastructure are considered and an optimal solution is chosen. The chosen infrastructure (more specifically Amazon Elastic MapReduce) removes the need of dedicated personnel for provisioning of resources because of the EC2 options for renting clusters. The processing of the relevant blips is done in a bulk by the Hadoop framework and thus achieving good performance of the system for that specific task. Within this paper also a Cassandra's database system is being considered for querying and access of recommendations for users.

DESIGN CONSIDERATIONS AND ALTERNATIVES

There are several key aspects to consider about the design. Since we will be proposing an infrastructure of a recommendation engine based on information scavenged from Tweeter, Google+, Facebook and other social networks we would need a way to extract, process, store and retrieve the information gathered for each user. Each blip consists of text, tags with structured pieces of information, the user, a location, references and others, thus we would need a database like system where each user has his/her own entry with his respective blips. So a sensible design decision is whether or not to use a structured parallel database or a cloud-based infrastructure for storing and processing this information. A parallel database approach would mean that our system would aim to improve its performance through parallelization of loading data, building indexes, evaluating queries [1]. It is more of a physical resource management system. Parallel databases improve processing input/output speeds by using multiple CPUs and disks in parallel whereas centralized client-server database systems cannot handle such applications. The strength of parallelism comes that many computational steps are performed simultaneously. Parallel databases can be divided into multiprocessor architecture where we can either have shared memory (multiple processors share the main memory space), shared disk (each node has its own main memory and they all share mass storage) and the other architecture group is the hybrid one (Non-Uniform Memory Access, Cluster). The advantages of such as structure would be a fast access time and queries, which our system needs. A cloud-based infrastructure is mainly a large number of computers that are connected through a real-time communication networks, hence a distributed system over a network. That would mean that it can run a program or application on many connected computers at the same time. This is achieved through virtualization of real server hardware. The

virtualization itself gives great scalability of the system and therefore in the problem at hand it is a great advantage to use a cloud-based infrastructure. The problem in parallel databases infrastructure is that this approach does not scale well if the information becomes too large (if our company's idea starts to prosper and many users start to use the service and there is a huge amount of new blips).The problem with a cloud-based approach would be how to store the processed blips and how to access them. That would mean that we would either need to store the information in files or in some databases in the cloud which would produce latency that might affect the performance of our system. Since in our system the processing of the blips does not need to be instant we can perform it in a bulk. Therefore we can consider the use of Hadoop , which is a software framework for storing and large scale processing of data-sets on clusters. Hadoop seems ideal for the job of post-processing since it has a great computational power and its weakness is to when the processing needs to be done in real-time and on smaller data, which is not the case here.

Another key decision in our design is that since our company is operating under a budget of about B million pounds we need to figure out if we should build our own private cloud or rent an existing provider. Because we would be working with an unknown number of blips (users' activities on the social networks vary) and we do not know how many users would subscribe for our service a sensible approach would be to rent an existing cloud for the period of two years. This would be the better approach because if we were to build our own cloud it might not be utilized or it might not be enough for the processing, storing and accessing we would need in our recommendation service. If we were to build our own private cloud for let's say half of the budged and leave the other half for personnel, maintenance and problems that might arise, we might come across that we utilize only a small part of our hardware or if we have too many requests and information flowing the bought hardware might not be enough and that would call for more money to be spent on extending the cloud. An alternative approach would be to rent an existing cloud service such as Amazon EMR. With it we can launch either a persistent cluster that stays up indefinitely (to store our blips and information) or a temporary cluster that terminates after analysis (processing the blips and recommendations) [2]. Amazon EMR has support of variety of Amazon EC2 instance types (standard, high CPU, high memory, etc.), therefore it can be used for many the requirements of our system. There are three options for pricing Amazon EC2: On-Demand, Reserved, and Spot. And when we launch a cluster we choose how many and what type of instances to provision. So in our system a better approach would be to rent let's say x instances for storing the processed blips and y instances for the processing of the recommendations. And since we need an elastic system the instances x and y can share their resources when one is idle or another has high load of tasks. Another advantage of renting is that we can reserve some number of instances, which is cheaper and useful to have as a backup plan in case we need further resources. This reduces the cost but makes the system more flexible and to scale better if high load occurs. We can rent on-demand instances for the processing of blips or storing of blips if all these instances are overloaded. Renting a cloud would provide better utilization of the system and also it would remove the need of maintenance of hardware and fits well with the unknown data-set we have.

SOLUTION

Our solution consists of a virtualized approach of the system for processing and storing of blips and a parallel database for storing, querying and accessing recommendations, hence a hybrid one. First we would like to rent a cloud. I have chosen the Amazon EMR service because of its flexibility for payment plans and renting instances. Second we would like a parallel database system of our own in order to deal with recommendations and querying for the end-users. We have chosen a hybrid approach in order to avoid the disadvantages of both types. More specifically the latency that might arise within the cloud-based approach would not be appropriate for the end-user interface and large computations of blips in a bulk can be done better with the Hadoop framework. The file system best suited for our problem would probably be therefore HDFS since it is more tailored for large-scale data-intensive analytics. It has monitoring, error detection, fault tolerance and recovery. Its files are large compared to normal standards and in our system this would be the case if we store each user's blips in a separate file. The database system that would be used in my solution is Apache Cassandra [3]. This is a good choice because we need a database system that would handle the queries described in our specifications and also when we need fast retrieval of computation results (derived recommendations for users). Since our system demands high availability and needs scalability this choice of a database is really good. Not only does it provide the previous mentioned benefits but it does it without compromising performance. Cassandra also supports replication across multiple datacenters and provides lower latency for the users. The replication ensures that our data would survive regional outages.

We need a way to monitor the above mentioned services for new blip. The new blips will be gathered considering the source service's API with an extra field in the blip's table entry for specifying it. Other fields that the blips would consist of are the tag field that would act as a unique ID for each blip. A simple scheme of the monitor service getting the new blips is shown is figure 1. The retrieval of the new blips would be done in the cloud infrastructure with the help of the Hadoop framework. Since in our specifications we need to get rid of duplicates that can arise within blips we can use MapReduce to remove the duplicates. In our specifications it is mentioned that such deduplications need not be done in real time so we can use a set time interval after which the MapReduce is implemented and in this way we can use it over a large data-set of blips. This way we utilize Hadoop's computational advantages and achieve better performance. The blips that already come with a tag (value) like tags in tweets and other social media are stored directly as a blip-tag

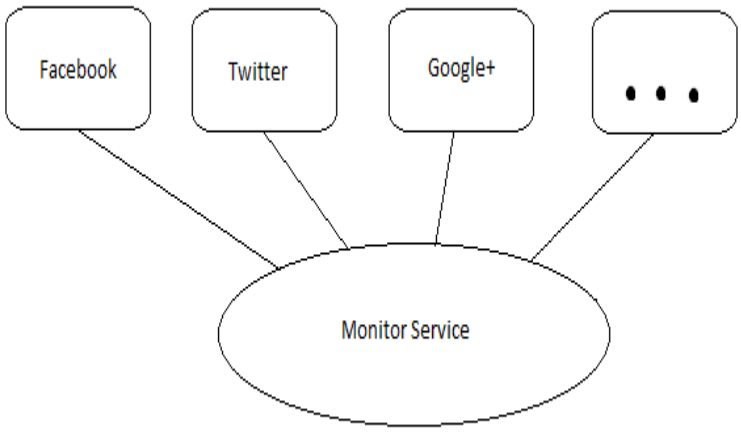

Figure 1. Monitor retrieving new blips from social media

tuple and are called explicit tags. The other type (implicit tags) is done by processing information about the user and the location information gathered when receiving the new blip. The will be the result of the post-processing done by the MapReduce in the Hadoop framework. In our specifications it is said that these computations are not needed right away so we can do this processing once every n minutes in a bulk in order to exploit Hadoop's processing power more efficiently. In order to avoid latency the number n would not exceed sixty and would be scaled according to how many new blips arrive. A scheduling algorithm might be implemented to avoid starving the post-processing task. For example if we have a high load of new blips n would decrease and if there is little load n would increase. In such implementation if little load is encountered the processing resources will be allocated and used by a different process and if high load happens more instances are sought to deal with the post-processing.

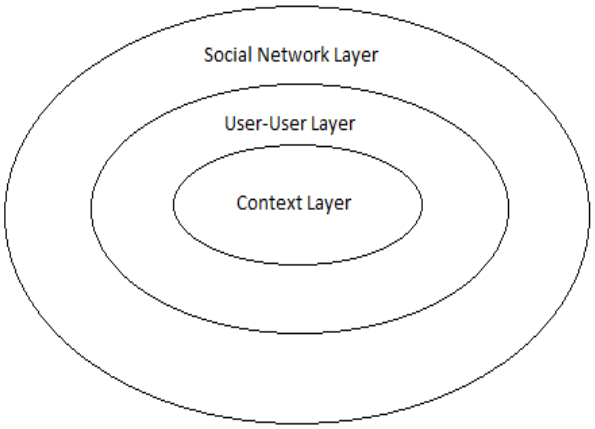

Figure 2. Layering of the User network

Since in our system each user would have its own file consisting of his blips and tags we can correlate each user with other users by a Layer based approach (figure 2). This approach is an abstract way of relating the users to networks based on their blips. For instance a way to relate users to a subnet would be by social media they share (twitter, facebook, google+) or we can construct several layers of such subnets. If we choose to do a layer approach as to the top layer being their social media, second layer (User-Users Layer) - other users referenced in their posts/tweets, or friends (Basically information that might relate users to one another that has been processed from their blips and tags), the third level would be relating the users based on the context of their blips' context (common tags, similar interests based on post/tweet content). This layering would be done by data mining algorithms that would be run on the servers as a type of post-processing of the information from each user's file.

Aggregation among multiple dimensions would be done within our database system (Cassandra). This database scales well with large amount of data and when the computations are performed in a bulk it would not hinder its performance. It is easy to group tags by location or locations by tags within our database since it is simply a sort by query with respect to the dimension wanted. The other advantage of such approach is that the results once reached can be stored in an easily and afterwards accessed/retrieved quickly. In order for our recommendation system to be effective this is a key factor since the retrieval of recommendation need to be fast in order for our customers to be satisfied.

We also have a requirement to detect trends and have a notion of tag disambiguation. In order for our system to do that we need a tool for semantic checks that would derive such assumptions based

on the content (text) of our blips. Such tools exist and can easily be implemented within our cloud. My choice to tackle this requirement is the following. We are to use Hadoops framework to implement a MapReduce job in java that does the semantic analysis over our blips. These should remove some disambiguation of our tags, and after such analysis our data will be sorted in semantic clusters. Such clusters would be what our blips are referring, whether it is a product, an event or a movie. These clusters however are only an abstract idea. In reality after the semantic analysis are done we would add some references to each blip in our database. These references would be mainly keywords with which when a query with other relevant keywords is executed it would find the blips that belong to the same cluster center. More specifically these clusters are keywords that grasp one context meaning and refer to the similar activities such as -watch, movie, TV, cinema. They would refer to a cluster center of something to do with watching movies. Relevant recommendations to such a user would be Movies or TV series.

The messaging part of our system in my opinion would be better to be via an email client that might be easily implemented on one of our rented servers or we can even rent a domain for an e-mail client itself. The other part of this requirement is what information would be used to alert the user. It will be achieved by using standard data-mining methods on open-source databases one the web. And these methods would be implemented with relevance to the interests that have been derived for our specific user. An overnight email would be the best approach for such messages because such data-mining techniques require a lot of processing time in order to be accurate and hence overnight would allow them that free time to do them. If we were to do the in real-time then we would need to consider a database approach that would have an enormous data-sets and would be highly computational so it would have latency which we do not want in our case (imagine if a user receives a message about a theatre performance that he wanted to attend an hour after it has started). So after having our data-mining recommendation generation service for users' interests and our messaging system we can implement similar data-mining techniques to derive product/event/other users recommendations. The only difference would be the data-sets we are doing the analysis on.

Another useful feature of the Cassandra database is that it supports timestamp column entries such as when a blip is inserted within our database we can use this time of the insertion as a timestamp. So we can easily implement temporal locality within it. According to the timestamps of the different blips we can make a priority to note the importance of the tags. In other words we can make sure that recently used tags have more importance by having a priority factor like an integer variable that starts with 100 and decrements each day until it reaches 0 (after several months if we don't want to keep outdated information or never if we do) and when it does is not considered in our recommendation generation at all. In this approach we can add a weight vector when analyzing the blip information with accordance to this number. In which case the lower the vector the less the information would affect our recommendation.

The implementation of such a hybrid system between parallel databases and virtualization provides us with an infrastructure that has availability (Operates about 99.999% of the time), which is ideally what we want. But we also want it to be elastic so that its resources should be dynamically allocated. And parts of our system should not be idle while other parts run extensive computations. That

is ideally achieved with a good notion of load balancing. Since our system is heterogeneous we will use dynamic load balancing with the help of Elastic Load Balancing that is implemented within the cloud we would rent. It automatically distributes the incoming traffic/computations across multiple Amazons EC2 instances [4]. It provides us with even greater fault tolerance while balancing the capacity of the instances used. It detects unhealthy instances and reroutes the traffic to healthy instances until the unhealthy ones are restored. We can ultimately achieve a dynamically balanced resources distribution across our instances in order to take into the account high and low loads of our blips. Our system will scale based on the load either by reallocating resources from different instances or by freeing them to be used by another process of our system.

CONCLUSION

As a conclusion we shall summarize how our system design meets the specifications set out by Massbase.

Monitoring the services for new blips and removing duplications is implemented with Hadoops framework and the notion of explicit and implicit tags is introduces as a tuple of blip-tag that in the first case is about an already known tag and does not lose its connection to the blip and in the second its tag has gone through post-processing using Hadoop MapReduce techniques.

The idea of how to identify networks of users is provided by a layer-type system consisting of three layers (Social, User-User and Context Layers) and because of its simplicity it can even be extended to even more layers.

Multiple dimension aggregations are provided by our system thanks to the Cassandra's database approach that we use for our front-end providing fast retrieval and efficient storage of recommendations.

Our system identifies information in the blip with the help of semantic tools implemented in the Hadoops framework and sorts this information by keywords that can later be used to access relevant information from our database. Data-mining open-source databases with accordance to a user's interests are suggested and an over-night messaging client solution is proposed. Also similar data-mining techniques are considered for products/event recommendations.

Temporal locality is tackled by using a priority based system on the timestamps of blip entries in the database. Availability is provided by the fault tolerance of Cassandra and Amazon's rented cloud distributed systems. Load balancing is achieved by Amazon's Elastic Load Balancing service implemented in the cloud-based infrastructure. Finally we can safely assume that the provided solution would scale to infrastructure of any size because of the scalability of our cloud-based infrastructure and Cassandra's database system used within it.

REFERENCES

[1]. http://www.cs.berkeley.edu/~brewer/cs262/5-dewittgray92.pdf

[2]. http://aws.amazon.com/elasticmapreduce/#pricing

[3]. http://cassandra.apache.org/

[4]. http://aws.amazon.com/elasticloadbalancing/