1.

There are quite differences between relational database and NoSQL database. Actually, in my view, NoSQL database seems like a derivative of relational database with more availability and flexibility. Relation database is more suitable for many business applications in finance, accounting, and enterprise resource planning as a result of its normalized data characteristic, which means data is stored in tables of RDBM, and a table is a collection of related data items, including columns and rows. These databases do need a predefined schema, all columns and data type should be known in advance, so that data can be stored into the database. However, NoSQL database will dynamically define data model which leads to an important fact that NoSQL database can take graphs, column-oriented, file-oriented as key, value pairs.

Let's put it in another way, relational database is more likely to be used for applications who need stable data, service, and consistency of data type. NoSQL database is suitable for the companies who need fast creativity and keep agility. For example, some online video games and E-commerce web applications. The applications with large amount of data, low latency and fast response are the first choice for them.

Referring to my own experience, one of my project's tasks is to build a door control system base on face recognition technique. This required me to store massive face pictures as resources for recognition, at that time, I chose the Neo4j (which is one of the NoSQL databases) as my database. Compared with most relational database, it has a great capacity of data, low latency, and faster response advantages which dramatically speeded up of recognition process. This is also because with the amount of data or traffic increases, the advantages of horizontal expansion architecture database are particularly obvious.

The relational database cannot realize horizontal expansion is because this type of database must keep consistency which means the users should see the newest data when they are looking at the same database, however, after changing the nodes and updating databases between several computers, there will be latency causing this problem.

The relational database also reflects a problem that there is a bottleneck under high concurrency, especially in the case of frequent writing and updating. The bottleneck results in errors such as high CPU workload of database, slow speed of execution of SQL, insufficient connection reported by clients. In contrast, the ElasticSearch, a representative product of NoSQL, is aiming for solving the problem of weak full-text searching ability of relational database by reverse index.

As far as I know, NoSQL database can be divided into four main types, including key-value, document, graph, and wide column. In conclusion, relational database is supposed to store some certain types of stable data with understandable, simple operations, while the NoSQL is pursuing for more flexible, faster, diversified implementations, it is to say that you can store any types of data as you want, you are able to dynamically define data model.

2.

First of all, Hadoop and Spark both are big data architecture, but they exist for different purposes. Hadoop is essentially more of a distributed system infrastructure, it allocates huge data sets to multiple nodes in a cluster composed of ordinary computers for storage, which means that you don't need to buy and maintain expensive server hardware for storing data. At the same time, Hadoop also will index and track these data, making the efficiency of big data processing and analysis reach an unprecedented level. However, the Spark is an available tool used to process the data stored in the distributed server, and it doesn't store the distributed data.

Hadoop is a distributed system infrastructure developed by Apache Foundation, Hadoop implements a distributed file system called HDFS, HDFS is characterized by high fault tolerance and is designed to be developed on inexpensive hardware. Moreover, it provides high throughput to access application data, which is suitable for applications with very large data sets. The core design of Hadoop framework is HDFS, MapReduce, and YARN. HDFS provides storage for massive data, YARN is a general resource management framework of Hadoop, which is used to allocate and manage resources for application running on the cluster, MapReduce provides calculation for massive data.

As for Spark, it is a fast and universal computing engine designed for large-scale data processing, which means Spark is more or less similar to MapReduce in Hadoop. It has main advantages of MapReduce. Spark's output results in the middle of task can be stored in memory, so that there is no need to read and write them in HDFS, this fact causes that Spark's performance and operation speed are higher than MapReduce's.

To be more specific, Hadoop's MapReduce is more likely to process data step by step, it firstly read data from disk, after finishing the process, it will write result into disk again. Then read the updated data from disk again, processing again, writing again…. It's just like a circle, it takes a great amount of time to spend on reading and writing between disk. It will affect the processing speed. In contrast, Spark reads data from disk, and puts intermediate data into computer's memory, completes all necessary analysis and processing, and it will write the final result into cluster, so Spark is faster than MapReduce to a great extent.

When it comes to fault tolerance aspect, as a result of that MapReduce will store data into disk after processing, there will be some severe situation like power failure or other natural disasters causing data lose errors. However, the Spark's data objects are stored in the Resilient Distributed Datasets RDD, which is a set of read-only objects distributed in a group of nodes, if there is a part of data set lost, they can be reconstructed according to the data derivative process. Furthermore, when RDD is calculating the Checkpoint can be used to realize the ability of fault tolerance.

The relationship between Hadoop and Spark actually is tight to some extent, Hadoop provides HDFS for achieving distributed data storage function as well as MapReduce for processing these data. So that Hadoop is not necessary to rely on spark to do processing data. Of course, Spark also can use other distributed file systems instead of HDFS. But they also totally can work together, Hadoop offers distributed clusters and distributed file system, Spark can rely on the HDFS of Hadoop instead of MapReduce to make up for the lack of computing ability.

So, when should we choose Hadoop or Spark? As we mentioned before, MapReduce is distributed computing framework, it's running on the YARN, and collaborate with HDFS for distributed data computing. Spark is more likely be a distributed computing framework like a equivalent to a improved version of MapReduce, and it supports memory-based iterative computing, In most cases, clients are prone to choose use Spark to process data on HDFS and work with Hadoop.

From this point of view, if there is only massive data storage need, it is no doubt to choose Hadoop since Hadoop's MapReduce is regarded as a distributed storage standard in the industry, while Spark barely can be used for running computing instead of Hadoop. Hadoop + Spark is the first choice when it comes to data processing on HDFS. Compared with MapReduce, using Spark to process data not only can improve the performance by more than 10 times, but also Spark's RDD has related APIs are pretty rich and can support SQL to process data.

3. Map-Reduce-Hadoop

First, the paper is mainly addressing the problem of speculative execution of Hadoop. Its solution is based on analyzing and identifying errors both in the threshold-based scheduling algorithm in Hadoop and with progress-rate-based algorithms in general. After that the author decided to focus on a more scalable, robust scheduling algorithm, which is called LATE Longest Approximate Time to End), to address this problem. The LATE algorithm plays a great important role in improving the response time in Hadoop by using estimated finish times to respectively run the tasks that damage the response time the most. With lots of experiments and convincing comparisons, it depicts that LATE has a better performance than Hadoop's fault speculative execution algorithm in real workloads on Amazon's EC2.

LATE's major method is based on estimated time left, and it also can monitor the slow task in advance. Apart from that, there are also other parameters like SpeculativeCap, SlowNodeThreshold and SlowTaskThreshold. Properly adjusting and setting these parameters also can ensure reasonable behavior. In my opinion, there are four valuable points in this paper that we should pay our attention as well as in the future task-solving progress, the first one is that should not wait to base decisions on measurements of mean and variance, in other words, we have to make decision early. The second one is that when we are speculating tasks, we should use finishing times instead of progress rates. The next one is nodes are not equal, avoiding assigning speculative tasks to slow nodes. The last one is that we must appropriately set Caps, which is used to guarantee system won't be overloaded.

The main problem, which this paper is trying to address is that the Hadoop's speculative scheduler is not enough to be sufficient. And it is really a complex issue since speculative tasks compete for certain resources like network bandwidth. In addition, it is also important to choose which node to do speculative work. Lastly, in a heterogeneous environment, stragglers are quite independent as early as possible to impact reduce response times, and it is too difficult to tell between nodes that are slightly slower than the mean and these stragglers. So, the LATE, which is proposed in this paper, is quite worthy and valuable as it not only try to prioritize tasks to speculate, select the fastest node to run on, but also use cap speculative tasks to prevent trashing to a great extent. As a result, this method has a great positive impact improving response time of MapReduce work by a factor of 2 compared with the previous method.

Author explains the rationale of speculative execution in Hadoop in the first part of the paper as well as some assumptions to enable us to know why previous speculative execution is not good enough and breaks down sometimes. For example, Hadoop holds the view that every slow node has a problem, however, there are other reasons to lead to this problem like multiple generations of hardware or multiple virtual machines running on each physical host causing heterogeneity and so on so forth. And there will be contention between nodes for computer resources (like network bandwidth mentioned above).

So, how does the LATE work, it estimates the progress rate of each task as ProgressScore/T and predicts the time to completion as (1- ProgressScore)/ ProgressRate.

And its algorithm works as follow:

If a node asks for a new task and there fewer than SpeculativeCap(mentioned above) speculative tasks running, then only select the tasks which are above SlowNodeThreshold, this is because that author wants to launch speculative tasks on fast nodes to see how better this algorithm is. But I think there should also be tests on the slow nodes, stragglers as comparison to see this method is good enough in any situation. Then these tasks will be ranked instead of being speculated by estimated time left. Finally, node will choose the highest ranked task to execute. I also noted that there are some confusing points in the following content, author said the best setting for SpeculativeCap, SlowNodeThreshold and SlowTaskThreshold are 10%, 25th percentile, 25th percentile separately. But it isn't suitable for all works, only for the experiment of this paper. Besides, as for data locality, author only assumed that it's because most maps are data-local, network utilization during the map phase is low. But I am afraid that without the local copy, the processing of this work will lack of instability.

After illustrating how this means works, author use a section to conclude some advantages of it. It is robust to node heterogeneity. It takes into account node heterogeneity when deciding where to run speculative tasks and focus on estimated time left rather than any slow tasks as well. However, there is a severe problem that the heuristic maybe cannot incorrectly estimate if a task which was launched later than an identical task will finish earlier. It might be not a common and regular situation to happen, but it does influences the performance of Late to a great extent.

The next section of this paper is that evaluation of the LATE. There are two environments: large clusters on EC2 and a local virtualized testbed. The first testament of EC2 is driven by the Amazon since these experiments were exposing a callability bug in the network virtualization software running in production that was causing connections between their VMs. In all tests, they configured the distributed system to maintain two replicas of each chunk as well as each machine running 2 mappers and 2 reducers. Firstly, they used plenty of graphics and tables and tested the heterogeneity on EC2 since I/O devices are shared between VMs. There are some problems, for example, the results are not inapplicable to CPU as EC2 tends to try allocating a user's virtual machines on different physical hosts. When it comes to contention on I/O performance, it appears that there are few extremely slow stragglers, they don't know the cause of these. In my view, I reckon that faulty and mistakes of machine could be a factor somehow. But it shows that the I/O performance of small VMs is better than large ones.

The experiment of impact of contention at the application level is sorting 100GB of random data using Hadoop, the speculative execution disabled as for best performance, while I think there also should be control group in case there are some extreme situations which will lead to needless pay. Before experiments efficiently guarantee the side effects and after these they conducted scheduling experiments on EC2. There are two settings: heterogeneous but non-faculty nodes and an environment with stragglers, author took several aspects to see how goo the LATE is, they are scheduling in a heterogeneous cluster, scheduling with stragglers, difference across workloads. The results of first two respects both are showing the LATE finished these experiments faster than Hadoop's native scheduler and no speculation under worst-case, best-case, and average case. Author also noted that there is a "lucky" element involved in the differences across

workloads tests, if there are a data chunk's two replicas both happen to be placed on stragglers, then no scheduling algorithm can perform well, since this chunk will be slow to serve. As for second test local testbed experiments, it's quite similar to previous experiments, and results also show the LATE extraordinary performance on the tasks compared with native speculation and no speculation.

Following that, author did sensitivity analysis in three aspects, SpeculativeCap, SlowNodeThreshold and SlowTaskThreshold, trying to find out these parameters' stability and what is the best setting normally. One problem right here is that author adopted a synthetic sleep workload. However, sleep workload does not penalize the scheduler for launching too many speculative tasks since sleep tasks do not compete for disk and network bandwidth. In my perspective, SpeculativeCap more and less will be influenced since it is supposed to prevent needless speculative tasks. Nevertheless, the result of analysis of sensitive SpeculativeCap is that after some minimum threshold, response time stays low and wasted work does not increase greatly with controlling other variables. This is a pity that the optimal values for SpeculativeCap in sensitivity experiments and evaluation are quite different, however, author cannot run EC2 test cluster experiments as a result of losing access to the test cluster. We also see the other two of them have great stability and as long as SlownNodeThreshold is higher than the fraction of nodes that are extremely slow or faulty, LATE performs well anyway.

All in all, under the situation of increased interest from a variety of organizations in large-scale data-intensive computing and the advent of virtualized data centers, as well as heterogeneity will be a problem in private data centers as multiple generations of hardware accumulate and virtualization. These aspects mean that coping with stragglers' affects on MapReduce workloads will be a increasingly important problem. The Hadoop's native speculative scheduler is able to fail in speculative ways. Another method like adopting the mean and the variance of the progress cannot identify slow tasks eventually not enough. All of these lead to the need and advent of a new available approach- LATE. Its goal is to identify the tasks that will damage response time the most, and do so as early as possible, rather than waiting until a mean and standard deviation can be computed with confidence. It is a really reading-worth and sophisticated paper to illustrate us a totally new and approach to address the problem encountered by all the computer field, even though there are still some confusing points need us to explore and pay attention.

References

[1] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In Communications of the ACM, 51 (1): 107-113, 2008.

[2] Hadoop, http://lucene.apache.org/hadoop

[3] D. Gottfried, Self-service, Prorated Super Computing Fun, New York Times Blog, tinyurl.com/2pjh5n

[4] Figure from slide deck on MapReduce from Google academic cluster, tinyurl.com/4zl6f5. Available under Creative Commons Attribution 2.5 License.

[5] R. Pike, S. Dorward, R. Griesemer, S. Quinlan. Interpreting the Data: Parallel Analysis with Sawzall, Scientific Programming Journal, 13 (4): 227-298, Oct. 2005