Tathagata Das [email: tdas@eecs.berkeley.edu]

Biography and Research Interests

I am a third year graduate student in CS, working in the AMP Lab (Soda Hall). My research interests include datacenter frameworks for processing "big data". My current research focus is in building a framework for large scale processing of streaming data. In this course, I would like to learn more about parallel processing techniques in order to further optimize datacenter frameworks.

Parallel Application

Spark Streaming – large scale near-real-time processing of streaming data

[This is not a real application, but a framework for writing Java-based cluster applications. I hope this works for the purpose of this homework.]

Much of "big data" is received in real time, and is most valuable at its time of arrival. For example, a social network may want to identify trending conversation topics within minutes, an ad provider may want to train a model of which users click a new ad, and a service operator may want to mine log files to detect failures within seconds. To handle the volumes of data and computation they involve, these applications need to be distributed over clusters.

There has been substantial work in large-scale batch processing systems like Google MapReduce [1], or Yahoo Hadoop [2]. In such systems, a job to process a large data file is divided in to small deterministic tasks – map tasks and reduce tasks. Each map task is designed to process a pre-partitioned block of the data file stored in a distributed file system running on a cluster of nodes. The map task processing a block is usually run on the same node that has block in its local file system, thus ensuring data locality and avoiding transfer of the data across the nodes. Only between the "map" and "reduce" stage the data is "shuffled" between the mapper nodes and the reducer nodes. Finally the result of the reduce tasks is written back to the distributed file system. In recent years, there has been various optimization of the basic map-reduce. For example, the Spark [3, 6] project (developed in AMP Lab) allows execution of a generalized DAG-of-operators and allows intermediate results to be cached in memory for repeated processing. It provides a convenient abstraction of a Resilient Distributed Dataset or RDD [3], which is a fault-tolerant, immutable collection of data that is distributed over a cluster of machines. It also provides a functional API of operators (map, filter, reduce, etc.) that is used to transform one RDD to another. This abstraction allows a sequential-like Spark program to automatically process large data distributed across a cluster (kind of like OpenMP's compiler directives automatically parallelize tasks across processors).

However, there has not been enough work done in developing a framework for processing large data streams with low latency. Nor have there been good abstractions for expressing distributed stream computations. Existing stream processing systems process data in a cluster of nodes in the record-at-a-time model. Each node in the cluster is assigned a specific operation. As each record arrives to a node, a mutable state is updated, and the transformed record is forward to the next node in the pipeline. Even though such systems achieve low latency (order 100ms), they do not have great fault-tolerance properties. Existing fault recovery techniques are either inefficient (replication of nodes with fast failover and 2x hardware cost) or slow to recover (replaying unprocessed records to a single replacement node).

This led us to the on-going Spark Streaming project [4, 6]. In this project, we propose to express a computation on live data streams as a sequence of batch computations on small batches of data. Since batch computations are composed of deterministic tasks, they have efficient fault-tolerance techniques – failed tasks/lost intermediate data (due to node failure) can be easily recomputed in parallel on multiple nodes. This allows stream processing to share the benefits of batch processing model – processing on a large clusters with fast, efficient parallel recovery from failures. Notably, this comes at the cost of latency - the end-to-end latency is at least the size of the batches that we can process. For certain workloads, we have been able to achieve sub-second latencies while processing data over a 100-node cluster. Furthermore, similar to Resilient Distributed Datasets in Spark, we provide the abstraction of Discretized Streams [4] that allows streaming computation to be expressed very conveniently. We have implemented complex machine learning algorithms like MCMC-based traffic transit-time estimation algorithm on Spark Streaming and have found it to scale to more than 80 nodes [5].

While it solves some of the problems that prior system had, there are a number of interesting challenges that are still unsolved.

(i) Imagine a computation where new data is received, partitioned by a primary key, and joined with existing “state” data to update it. For maximum efficiency, for every batch, it should be ensured that the partitions of the new data be located on the same node as the corresponding partitions of the state data with which they will be joined. This comes at the cost of load-balancing – one node having a large partition will always be overloaded thus slowing down all the processing. It is non-trivial to optimize this.

(ii) Such systems are fundamentally based on the MapReduce model, where there is a logical barrier between the “map” and the “reduce” stages. Any “straggler” node (node consistently running slower than average) and/or skew in the task computation times can easily delay the reduce stage, leading to lower under-utilization, lower throughput and/or higher latency. There may be many ways to address this - pipelining multiple batches to fill in the idle periods, etc. It remains to be tested.

Spark Streaming is publicly available as an alpha. It is distributed as a component of the Spark project (main website, Github repo). Feel free to download it and take it for a spin for your big data streaming applications.

Further resources on Spark Streaming: paper, talk, programming guide

References

[1] J. Dean and S. Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters.* OSDI 2004

[2] Hadoop - http://hadoop.apache.org/

[3] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael Franklin, Scott Shenker, and Ion Stoica. *Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing.* NSDI 2011.

[4] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker and Ion Stoica. *Discretized Streams: A Fault-Tolerant Model for Scalable Stream Processing.* UCB Tech Report.

[5] Tim Hunter, Tathagata Das, Matei Zaharia, Peter Abbeel, Alex Bayen. *Large Scale Estimation in Cyberphysical Systems using Streaming Data: a Case Study with Smartphone Traces.* ArXiv.

[6] Spark – http://www.spark-project.org