Paper Review

Bharath Venkatesh
bharat

MapReduce: Simplified Data Processing on Large Clusters

Summary

The core idea of this paper is to introduce a programming model which exploits parallelism to handle large data sets in distributed system. It represents data as pairs of keys and values. A *'map'* operation is performed on the input key-value pairs to generate a set of intermediate key-value pairs. These values are then subjected to a *'reduce'* operation which generates the resultant key-value pairs. The model uses a master-slave mechanism wherein the master schedules map and reduce tasks to its slave nodes which in turn compute and the place the result on the distributed system. It handles worker failures by maintaining the state information of each of its worker nodes and periodic pings to each of these nodes. It also takes into account that some worker threads might take very long to complete their execution thereby stalling further execution of the request, by using backup execution mechanisms towards the end of the query processing. The model is designed to conserve network bandwidth by making sure that the map and reduce tasks are assigned on nodes with close vicinity to the input data. The paper provides a detailed explanation of the experiments carried out to evaluate the model in terms of normal execution, no backup and node failures with respect to grep and sort programs. It also elicits first hand experience of this model in use at Google across a wide range of domains.

Pros: Why the Paper Should be accepted

- The key take away from this model is the fact that it provided a simple and powerful interface that enables *automatic parallelism and distribution* of large scale computations across large clusters of commodity PCs.
- Secondly it provided optimization techniques to reduce the amount of data sent across the network through the use of *network locality.*
- Appreciably, the model has efficient mechanisms to handle *slave failures and delayed workers* through the use of state information.
- There is *no constraint on the input* data format as long as it can be reduced to MapReduce key-value pairs. This makes the model applicable for several applications.

Cons: Why the paper may be rejected

- The model might not be suitable for *time critical applications* as it does not express an upper bound on the time it would take to finish the operation.
- Another key drawback of the algorithm is that, MapReduce leads to significant *memory consumption* due to fact that it needs data to be stored in the intermediary and it requires states of each node maintained on the master.
- Network locality assists in efficient bandwidth management. However the downside of using this approach is that it might *load* few of the slave nodes *unfairly* over other nodes based on input data location.

- Although MapReduce accounts for worker node failures, if master node fails it brings the entire application to a halt, thus leading to a *single point of failure*. Also a discussion on how master node is chosen initially and in the event of failure would have made the discussion more lucid.

Ideas for Future Research

- An interesting improvement over the existing approach would be to build a model that accounts for both *network locality* and efficient *load distribution.*
- Algorithms above the MapReduce model to counter for the fact that it cannot handle *multiple map-reduce requests concurrently.*
- *Mirroring Master* nodes will further make the system more robust to failures and will reduce the downtime during such scenarios.

SPADE: The System S Declarative Stream Processing Engine

Summary

The paper presents SPADE, a declarative stream processing engine developed by IBM as a middleware for large-scale distributed systems. The system has three components: (i) An *intermediate language* that allows for flexible composition of distributed data-flow graphs. (ii) Supports basic stream relation operators and user defined operators through generic *stream processing operators*. (iii) Stream *adapters* to publish and consume data from external sources. The system is expressed as a data flow graph consisting of processing elements which are connected through streams. The *Dataflow graph manager* is responsible for matching stream descriptions of the output ports with the flow specifications of input ports. The *Data Fabric* establishes a transport connection between PE's and moves stream data objects from producer PE's to consumer PE's. The Resource Manager collects run time statistics for optimization. The PE Execution container provides a run time context to access the System S middleware. The paper also defines a *SPADE programming language* that hides the complexities associated with stream manipulations and application decomposition. It evaluates the performance improvements and scalability that SPADE can achieve by simulating a sample application for computing 'Bargain Index' in stock trading. The results show a *tuple ingestion rate* of 1.6 million tuples per second with application completion in less than 3.5 minutes.

Pros: Why the Paper Should be accepted

- One of the key advantages of SPADE is that it provided an *"on-the-fly processing"* model for processing data streams deviating from the traditional "store and process" model.
- Native support for edge adapters, toolkits for operators and the degreee of optimization it achieves implies that SPADE perfectly *suitable for high performance* scalable stream processing applications.
- SPADE allows for incremental development of code thus allowing a great deal of *flexibility and reuse.*
- The SPADE programming language provides a layer of abstraction, hiding complexities in computing infrastructure and alleviating the need to bother about transport issues between PEs'.
- SPADE provides seamless inter-operability with user defined operators, thus making it a plausible, customizable solution for future application requirements.

Cons: Why the paper may be rejected

- Although SPADE is proven to be efficient on the System S platform, its applicability is still limited in terms of *interoperability and deployment* across other platforms.
- SPADE is very limited in terms of the *customizations* that the user can specify. For instance if the user wants to specify optimization tweaks or control parallelization characteristics, such a capability is not yet present in SPADE currently.

Ideas for Future Research

- Refining SPADE to build a more *uniform platform* that is usable across multiple types of distributed environments
- Integrating *custom hardware platforms* to increase the speed of operation that can accommodate time-critical application.
- A *Pattern recognition* framework can be built above the SPADE platform to account for flexibility in user specification.