BDA5-8

1

Hello everyone, I am Haiying Che, from Institute of Data Science and knowledge Engineering

School of Computer Science, in Beijing Institute of Technology, in this session we discuss Distributed Graph Computing.

2

From the perspective of computing architecture, the data processing system is divided into data algorithm layer, computing model layer, computing platform and engine layer, etc.

Computing models are the way that different kinds of big data is processed in different scenarios, We have learned the batch processing, stream computing, **Large-scale concurrent processing model** for structured data, and In-memory database, in this session we discuss Distributed Graph Computing.

3

With Hadoop and Spark, we can accomplish data parallel processing and improve the computing performance. How about big graph that beyond the single node processing ability.

the graph parallel computing can solve this problem, represented by the Google Pregel, Apache Graph lab and Apache Giraph.

Pregel was first outlined in a paper published by Google in 2010.

It is system for large scale graph processing (think billions of nodes), and has served as inspiration for Apache Giraph, which Facebook uses internally to analyze their social network, as well Apache Spark’s GraphX library, which provides an API for Pregel computations.

The GraphLab project was started by Prof. Carlos Guestrin of Carnegie Mellon University in 2009. It is an open source project using an Apache License. While GraphLab was originally developed for Machine Learning tasks, it has found great success at a broad range of other data-mining tasks; out-performing other abstractions by orders of magnitude.

Apache Giraph is an iterative graph processing system built for high scalability.

For example, it is currently used at Facebook to analyze the social graph formed by users and their connections.

Giraph originated as the open-source counterpart to Pregel, the graph processing architecture developed at Google and described in a 2010 paper.

Both systems are inspired by the Bulk Synchronous Parallel model of distributed computation introduced by Leslie Valiant.

Giraph adds several features beyond the basic Pregel model, including master computation, sharded aggregators, edge-oriented input, out-of-core computation, and more.

4

Most of the graph processing algorithm can be expressed in terms of a combination of "traversal" and "transformation".

In the case of "traversal", it can be expressed as a path which contains a sequence of segments.

Each segment contains a traversal from a node to an arc, followed by a traversal from an arc to a node.

The main goal of Graph transformation is to modify the graph.

This include modifying the properties of existing nodes and arcs, creating new arcs / nodes and removing existing arcs / nodes.

The modification logic is provided by a user-defined function, which will be applied to all active nodes.

5

the most basic (atomic) unit is a "node" that contains its properties, outward arcs (and its properties) as well as the node id (just the id) that the outward arc points to.

The node also has a logical inbox to receive all messages sent to it.  
Pregel can be thought as a generalized parallel graph transformation framework.

6

The whole graph is broken down into multiple "partitions", each contains a large number of nodes.

Partition is a unit of execution and typically has an execution thread associated with it.

A "worker" machine can host multiple "partitions".

Let me briefly describe the process first:

Processing model is all active node will be executed whole processing completed when No more active node and No more in transit messages.

Superstep execution is, 1) Receive message from inbox, 2)Modify node and arc properties. 3)Halt self until new message received. 4) send messages to other nodes causing them active; 5)remove the existing or creating new arcs.

The execution model is based on BSP (Bulk Synchronous Processing) model.

In this model, there are multiple processing units proceeding in parallel in a sequence of "supersteps".

Within each "superstep", each processing units first receive all messages delivered to them from the preceding "superstep",

and then manipulate their local data and may queue up the message that it intends to send to other processing units.

This happens asynchronously and simultaneously among all processing units. The queued up message will be delivered to the destined processing units but won't be seen until the next "superstep".

When all the processing unit finishes the message delivery (hence the synchronization point), the next superstep can be started, and the cycle repeats until the termination condition has been reached.  
Notice that depends on the graph algorithms, the assignment of nodes to a partition may have an overall performance impact.

Pregel provides a default assignment where partition = nodeId % N but user can overwrite this assignment algorithm if they want.

In general, it is a good idea to put close-neighbor nodes into the same partition so that message between these nodes doesn't need to flow into the network and hence reduce communication overhead.

Of course, this also means traversing the neighboring nodes all happen within the same machine and hinder parallelism.

This usually is not a problem when the context nodes are very diverse.

In my experience of parallel graph processing, coarse-grain parallelism is preferred over fine-grain parallelism as it reduces communication overhead.

7

In each superstep, there are two phases which are compute and communicate.

In computing phase, the node finishes their computing task and in communicating phase, the vertices send and receive the message.

If vertices receive a message, it will be active in next superstep, if no message receive, the vertices will be inactive in next superstep.

8

The complete picture of execution can be implemented as diagram :

The basic processing unit is a "thread" associated with each partition, running inside a worker.

Each worker receive messages from previous "superstep" from its "inQ" and dispatch the message to the corresponding partition that the destination node is residing.

After that, a user defined "compute()" function is invoked on each node of the partition.

Notice that there is a single thread per partition so nodes within a partition are executed sequentially and the order of execution is undeterministic.  
  
The "master" is playing a central role to coordinate the execute of supersteps in sequence.

It signals the beginning of a new superstep to all workers after knowing all of them has completed the previous one.

It also pings each worker to know their processing status and periodically issue "checkpoint" command to all workers who will then save its partition to a persistent graph store.

Pregel doesn't define or mandate the graph storage model so any persistent mechanism should work well.

There is a "load" phase at the beginning where each partition starts empty and read a slice of the graph storage.

For each node read from the storage, a "partition()" function will be invoked and load the node in the current partition if the function returns the same node, otherwise the node is queue to another partition who the node is assigned to.  
Fault resilience is achieved by having the checkpoint mechanism where each worker is instructed to save its in-memory graph partition to the graph storage periodically (at the beginning of a superstep).

If the worker is detected to be dead (not responding to the "ping" message from the master), the master will instruct the surviving workers to take up the partitions of the failed worker.

The whole processing will be reverted to the previous checkpoint and proceed again from there (even the healthy worker needs to redo the previous processing).

Further optimization is available in Pregel to reduce the network bandwidth usage.

Messages destined to the same node can be combined using a user-defined "combine ()" function, which is required to be associative and commutative.

This is similar to the same combine () method in Map/Reduce model.  
In addition, each node can also emit an "aggregate value" at the end of "compute ()".

Worker will invoke a user-defined "aggregate ()" function that aggregate all node's aggregate value into a partition level aggregate value and all the way to the master.

The final aggregated value will be made available to all nodes in the next superstep.

Just aggregate value can be used to calculate summary statistic of each node as well as coordinating the progress of each processing units.

9

Many problems by nature form a graph such as the web, transport and social media. However, a common large scale distributed processing pipeline, MapReduce is ill suited for such tasks. MapReduce requires the data chunks to be processed independently. In other words, every job needs to know all the information to compute. This independence is certainly not the case with graphs and jobs might require previous computation or information from neighbouring jobs in order to calculate. To address this issue, Google’s Pregel architecture employs a message passing system creating a “large-scale graph processing” framework.

Another problem is the fact that MapReduce moves data around, for example by shuffling, in order to process it. This approach entails graph partitions moving around machines incurring high network overhead. A similar problem arises when MapReduce jobs are chained to implement iterative graph algorithms. Therefore, “MapReduce requires passing the entire state of the graph from one stage to the next - in general requiring much more communication and associated serialization overhead.”

Finally, “coordinating the steps of a chained MapReduce adds programming complexity that is avoided by Pregel’s iteration over supersteps”. MapReduce is not iterative, it can handle single iteration and requires the user to handle the iteration, i.e. the chaining of multiple MapReduce jobs. This situation can get complicated really quick for real world graph algorithms running at scale not only to implement but also to debug when things go wrong.

The user algorithm is distributed to many machines one of which becomes the Master. The Master partitions the graph into multiple graph partitions and assigns them to workers. A worker can have multiple graph partitions which contain a set of vertices and their corresponding outgoing edges which might point to any vertex in the graph, possibly stored on another worker.

Once the setup is complete, the master tells each worker to execute a superstep. At the beginning of a superstep, workers save their current state into persistent storage for fault tolerance. Workers then Compute() the user function for active vertices in their graph partitions sending and receiving messages asynchronously. When the worker is done computing, it tells the master it has finished and how many vertices will be active in the next superstep.

For message passing, 2 queues are maintained for superstep S and S+1 to simultaneously receive messages while computing the current superstep. Any message sent at superstep S is received at superstep S+1 to avoid locking at message level. There is no guarantee on the ordering of received messages. While computing, a worker can send messages to other workers including itself depending on where the vertex is stored. Messages are sent in a single network message when a certain message buffer size threshold is reached in order to reduce network traffic. If an optional combiner is specified by the user, messages can be combined when either outbound or inbound reducing network usage and storage respectively. For example, several messages can be combined into one if the user is interested in the sum of their value.

Aggregators can be specified by the user to collect results at each superstep. Workers provide values to an aggregator instance to produce a single local value Aggr.

At the end of the superstep, workers combine the partially reduced aggregator local values into a global value using a tree-based reduction and deliver it to the master.

When all workers complete a superstep, the master initiates the execution of the next superstep.

This process repeats until there are no active vertices.

The vertices become inactive after they vote to halt and there are no incoming messages to wake them up again.

When all vertices are inactive, the execution of the algorithm terminates.

Alternative to Message Passing

An alternative to message passing is a state passing model in which the sent messages are stored locally and only exchanged at the end of a superstep. In other words, the model latches onto the synchronisation barrier between supersteps to exchange the state of message queues rather than during computation.

As an immediate advantage, workers don’t incur any network traffic during computation. At the end of a superstep, messages can be sent in bulk, larger than the current buffer size, compensating for the handshake overhead of opening and closing multiple connections to different machines. In this case, all messages destined to another worker will be sent in one network connection.

A disadvantage would be a longer pause time in between supersteps. State passing approach would be worse whenever computation is less than information passing.

For example, if nodes only compute the average of neighbors, getting neighbor values outweighs computing an average causing the state to be transferred at the end of the first superstep. Perhaps, the user can specify depending on the algorithm when messages should be exchanged prioritizing computation or communication. In this manner, the message passing of the Pregel execution can be fine-tuned to optimize the performance of the user’s algorithm and dataset.

Scalability Limitation

**A common limitation** with large-scale distributed frameworks is the network traffic message passing creates. For dense graphs, the network could pose as a bottleneck when a lot of messages are passed during computation. Similarly, if there is a node with a high degree, whenever it sends or receives messages a single worker will have to deal with all the network traffic causing that worker to slow down and increase wait time on superstep barrier synchronisation.

**Another limitation** is the superstep barrier itself when every worker waits for other workers to complete to proceed to the next superstep. When scaled horizontally, the increased probability of failing machines might mean that the computation can wait a long time until all workers have completed and are ready for the next superstep.

For example, while a worker is being recovered another can fail causing another recovery during which the overall computation is stagnating. Unlike MapReduce in which other sub branches of the computation such as reduction continue, Pregel needs to wait for the barrier to be passed.

10

Applying the Data Algorithm Layer algorithms and combined with the Batch, stream, Massive Parallel Processing, In Memory Computing

Or Graph Computing model, we can process big data problems, but it is hard for us to implement all the algorithm and computing model by ourselves manually,

The Computing Platform & Computing Engine Layer can provide us the platform and engine which includes the needed tools, libraries to facilitate the implementing the complex Algorithms and Computing models.

**Computing platform and engine refer to a development integrated environment that provides Technical Standards, Computing Architecture, and a series of Development Technologies and Tools for big data computing and analysis.**

**The current representative computing platforms are: Hadoop, Spark, Storm, and Google's commercial platforms based on a series of big data computing technologies.**

11

Here in this table the computing model and platform are summarized, the computing pattern, representative products, storage system, Computing model, Computing platform and Key technologies

of Batch processing, stream computing, Massively Parallel Processing for structured data, in memory computing, Graph Computing and the interactive computing are showed in table.

12

In this session, we learned mechanism, principle of distributed graph computing and we summarized computing model. Thank you for your attention, if you have any question, feel free to contact me.