Harp project

Recent newly designed big data processing tools focus on the abstraction of data and related computation flows. For example, Hadoop MapReduce defines data as Key-Value pairs and defined computation as Map-Reduce tasks. Pregel/Giraph defines data as Vertices and edges in graphs and define computation as iterations of BSP fashion. Spark abstracts data as RDDs with transformation operations on top of them. However, communication patterns are not abstracted and defined in these tools. On the contrary, traditional distributed data processing tools represented by MPI, have abstractions on communication patterns which is called collective communication. But this kind of abstraction in MPI is limited. Collective communication in MPI is still based on arrays and buffers. There is no abstraction for more complicated data such as Key-Values, vertices and edges in tools mentioned previously. As a result, related communication patterns on these data abstractions such as shuffling on Key-Values or graph communication based on edges and vertices are also missed.

To improve the expressiveness and performance in big data processing, here I present Harp library, which provides data abstractions and communication abstractions on top of them. It can be plugged into Hadoop runtime to enable efficient in-memory communication to avoid HDFS read/write. Harp has the following features: hierarchal data abstraction, collective communication model, pool based memory management, BSP style Computation Parallelism, and fault tolerance support with check-pointing.

1. Hierarchal data abstraction

The data abstraction has 3 categories in horizontal direction and 3 levels in vertical direction (see Figure 1). Horizontally, data are abstracted as arrays, Key-Values or vertices, edges and messages in graphs. In vertical direction, there are two basic types at the lowest level, arrays and objects. At the middle level, they are wrapped as partitions such as array partitions, Key-Value partitions and graph partitions. At the top level, partitions forms tables. Tables are identified through table IDs. Partitions belonging to the tables with the same ID on different workers are considered to be one dataset. In this way, tables are seen as a distribution description of a data set. Collective communication operations are defined on tables and partitions.

1. Collective communication model

Collective communication are defined as movement of partitions within tables. Currently three different types of collective communication patterns are supported. The first type of collective communication are inherited from MPI collective communication operations such as broadcast, allgather. The second type of collective communication is inherited from Hadoop MapReduce communication pattern. That is regroup with combining and reducing support. Allreduce from MPI collectives is also implemented based on regroup + allgather. The third type of collective communications are abstracted from graph communication, such as sending messages to vertices and moving edges to messages. This type of collective communication is close to regrouping, but the regrouping destination is based on the distribution of another data set.

1. Memory management

Serialization are de-serialization are two expensive operations before data sending and receiving. Java default ByteArrayOutputStream is slow in serialization so that serialization and deserialization are done on byte arrays directly. Since byte arrays are frequently used in sending and receiving, they are cached in resource pool.

Figure 1. Hierarchal Data Abstraction



1. BSP style computation parallelism

Collective communication requires synchronization. Hadoop scheduler is modified to schedule tasks in BSP style.

1. Fault tolerance with check-pointing

Based on the scale of time length of execution, a jobs with large number of iterations are separated into a small number of jobs each with several iterations and submitted to the cluster.

Publication

Bingjing Zhang and Judy Qiu, High Performance Clustering of Social Images in a Map-Collective Programming Model, poster in proceedings of ACM Symposium On Cloud Computing, 2013