CME 323: Distributed Algorithms and Optimization, Spring 2016
`http://stanford.edu/~rezab/dao`.
Instructor: Reza Zadeh, Matroid and Stanford.

Lecture 10, 4/27/2016. Scribed by Ting-Po Lee and Yu-Sheng Chen.

10 Introduction to Distributed Computing

10.1 Communication Protocols

Several communication patterns exist between machines:

- All to one
- One to all
- All to all

We introduce their communication patterns and the associated communication costs.

10.1.1 All to one communication with *driver* machine

Computation is distributed among multiple machines and the results are sent to a single *driver* machine, as shown in Fig.1. Assume all machines are directly connected to *driver* machine the bottleneck of this communication is the network interface of *driver* machine. Let p be the number of machines (excluding *driver*), L be the latency between each pair of machines and the network interface of *driver* machine has bandwidth B. Assume all machines send a message of size M to the *driver* and *driver*'s network interface is saturated by every single message. i.e. machines queue up to send messages one at a time.

Each single message sent has cost:

$$L + \frac{M}{B}$$

Thus the overal communication cost is:

$$p\left(L + \frac{M}{B}\right)$$

10.1.2 All to one communication as *Bittorent Aggregate*

Another algorithm for all to one communication is known as *Bittorent Aggregate*. As in one to all communication computation is distributed among multiple machines but the results are aggregated through the communication between each pair of machines. The aggregation pattern can be seen as a tree structure or as depicted in Fig.2. Assume we are aggregating results from p machines, the results can be aggregated to a single machine in $\log_2 p$ rounds. Let L be the latency and B be

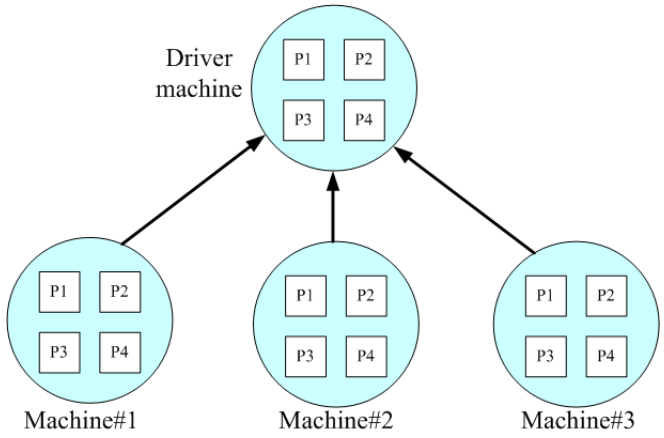

Figure 1: All to one communication with *driver* machine

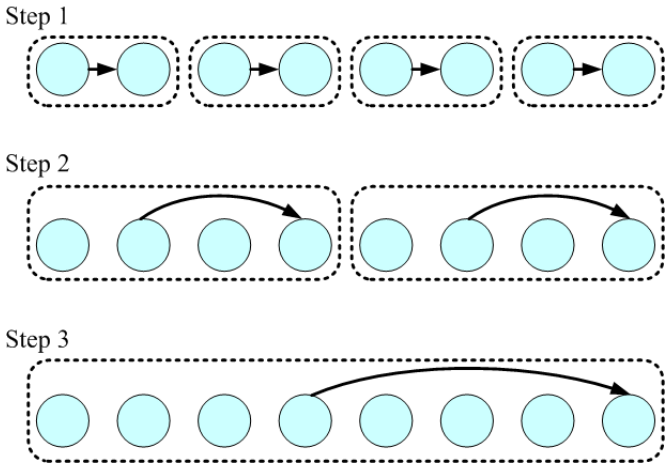

Figure 2: All to one communication with *Bittorent Aggregate*

the bandwidth between each pair of machines and in each aggregation round a message of size M is sent between machine pairs. The total communication cost is:

$$(\log_2 p)\left(L + \frac{M}{B}\right)$$

10.1.3 One to one communication

One to all communication has the same network configuration as all to one communication only with opposite data flow directions. The *driver* machine sends messages of size M to p other machines and *driver* machine's network interface is still the bottleneck of communication. Communication cost is the same as one to all communication with *driver* machine.

Also, we can borrow the concept of *Bittorent Aggregate* such that the message is relayed among machines in tree structure. Then the message can be spread among all machines within $O(\log n)$

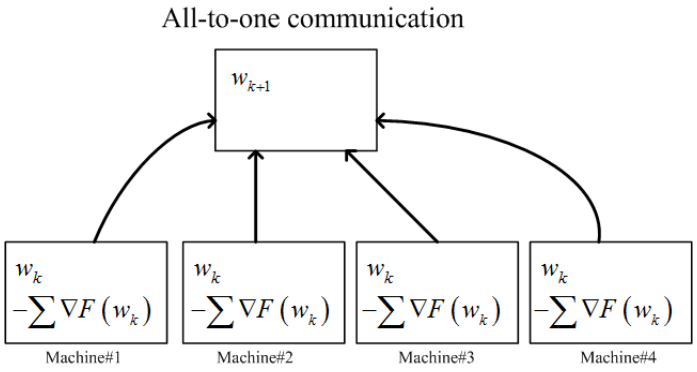

Figure 3: Gradient descent with distributed training data

rounds.

10.1.4 All to all communication

There are situations where all machines have to communicate with each other. Sorting is one example that requires all to all communication. Assume we have a large number of integers which is exceeds the storage of a single machine. The numbers are shuffled and distributed among multiple machines, each machine cannot determine the correct order of its numbers with communication with all other machines.

Some, but not all problems require all to all communication and it's valuable to find them. Relational database operations JOIN and GROUPBY are two other examples. In fact these two operations are implemented by sorting in many real world applications.

10.2 Gradient Descent

As described in previous lectures, many machine learning problems can be formulated as the following unconstrained optimization problems:

$$\min_w F(w) = \min_w \sum_{i=0}^{n} F_i(w, x_i, y_i)$$

We solve the optimization problem through gradient descent. In each iteration we have:

$$w_{k+1} = w_k - \alpha \sum_{i=0}^{n} \bigtriangledown_w F_i(w_k, x_i, y_i)$$

Where $w_k, w_{k+1} \in \mathbb{R}^d$, d is the number of parameters. Assume in training stage training data $\{x_i, y_i\}_{i=1}^n$ is distributed across machines as shown in Fig.3. Then in each gradient descent iteration, it takes one round of all to one communication to aggregate the value $\sum_{i=0}^n \bigtriangledown_w F_i(w_k, x_i, y_i)$ and another round of one to all communicate to broadcast updated w_{k+1}.

Here we list the communication terminologies usually used in distributed computing literature:

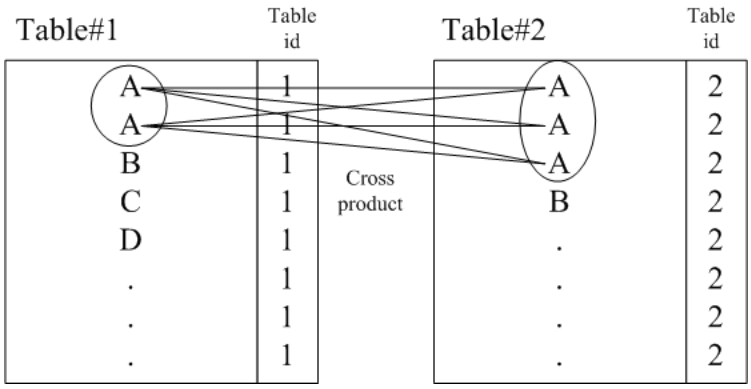

Figure 4: Relational JOIN

- All to one communication known as **Reduce**
- One to all communication known as **Broadcast**
- Reduce + braodcast known as **All reduce**

10.3 Relational JOIN

It is discussed previously that JOIN requires all to all communication and can be implemented as sorting. The algorithm is briefly described as follow. Assume two tables T_1, T_2 having rows $\{k_i, D_{1i}\}_{i=1}^{n}$ and $\{k_j, D_{2j}\}_{j=1}^{m}$ where rows are represented as tuple elements, k is the key we are joining on and D_1 and D_2 denote other columns of T_1 and T_2, respectively. We can then put the tuple elements of T_1 and T_2 in a single array A then sort with respect to k, elements with the same key k will be grouped together. Traverse the sorted array A and for each key k_i if we have elements from both T_1 and T_2, output all cross-products.

10.4 Midterm Review

10.4.1 Parallel Algorithms

By far, we have covered parallel algorithms for the following problems:

- Sum, max and $\oplus$ any associative operations
- Prefix sum
- Matrix computations
- Graph connectivity
- Minimum spanning tree
- Optimization

The algorithms above were discussed under PRAM model. Also, with communication protocols parallel algorithms for sum, max and $\oplus$ can also be considered in clusters.

In midterm problems, unless otherwise specified, one can assume set operation for binary search trees (BSTs) on PRAMSs.

10.4.2 Scheduling

Greedy algorithm for scheduling problem of minimum *makespan*. The derivation of greedy competitive ratio.

10.4.3 Brent's Theorem

Assume p processors, Brent's theorem states:

$$\frac{T_1}{p} \leq T_p \leq \frac{T_1}{p} + T_\infty$$

Note the lower bound holds even in clusters (where we have machines connected by network instead of processors in PRAM) but the upper bound does not hold in clusters.

Given p processors, we can sum an array of size n with the parallel sum algorithm discussed in class, which by Brent's theorem gives us an asymptotic upper bound of:

$$T_p \leq O\left(\frac{n}{p}\right) + O(\log n)$$

However, we can simply assign $\frac{n}{p}$ numbers to each processor and aggregate the results from each processor sequentially, which gives:

$$T_p \leq O\left(\frac{n}{p}\right) + O(p)$$

This simple implementation does not necessarily perform worse than the parallel sum algorithm which gives $O(\log n)$ depth.