CSE 550 – SYSTEMS FOR ALL
AUTUMN 2014
ASSIGNMENT 3: FINDING THE LEAST COMMON ANCESTOR
TEAM MEMBERS: ANTOINE KAUFMANN AND MAAZ BIN SAFEER AHMAD

1. ALGORITHM OVERVIEW

The first step in the process is to load the input files and generate a citation graph. Each vertex in the graph corresponds to a unique paper in the dataset. Similarly, each directed edge represents a citation with the source vertex citing the target vertex. Next, our implementation will proceed to compute the LCA for each pair of seed papers in two phases:

Phase 1: For each vertex, we compute its distance from each of the seed papers. We implement this step as a GraphLab vertex program, which will gather the distances from all incoming neighbors and combine those to find its own distance from each seed paper. This vertex program will be executed iteratively and asynchronously on all vertices, where a vertex will be "signaled" (for execution in the next iteration) by a neighbor whenever that neighbor's distance from at least one of the seed papers is reduced.

Phase 2: We combine the distance values calculated in phase 1 by folding over all vertices in the graph. For each vertex its distance for all pairs of seed papers will be checked, and if a shorter distance is found for a pair, this vertex will be considered the new LCA.

2. PERFORMANCE EVALUATION

For the performance evaluation of our implementation we investigated the following three aspects:

1. Scalability with increasing number of cores on a single machine
2. Scalability with increasing number of machines
3. Relationship of completion time to input size

NOTE: To limit the strain on the class EC2 budget we did not perform repeated runs of our experiments as would be required for a thorough performance evaluation. Therefore the following graphs will not include error bars and should in general be taken with a grain of salt.

2.1 SINGLE MACHINE SCALABILITY

Our first experiment was about quantifying the impact of the number of cores on the completion time for a fixed input size. Ideally the completion time should behave as O(1/n) where n is the number of cores. In our experiment we varied the number of cores by using different EC2 instance types: c3.2xlarge (8 cores), c3.4xlarge (16 cores), c3.8xlarge (32 cores). Note that the amount of RAM also varies proportionately with the number of cores. Smaller c3 instances did not have sufficient memory to run our implementation with the chosen input size of 500 seed papers.

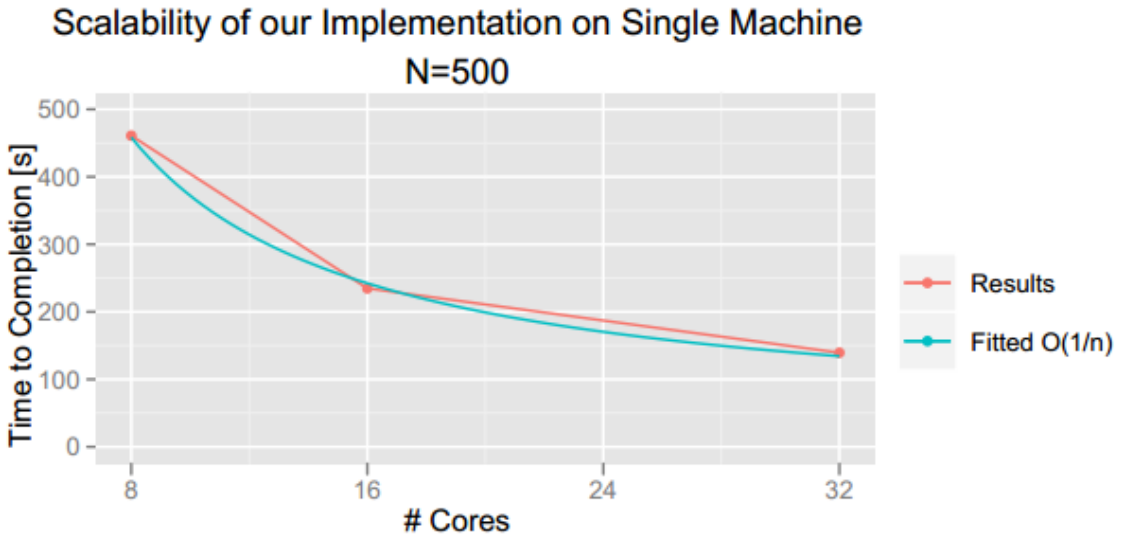

Figure 1. Scalability with increasing number of cores on a single machine

The results of the experiment are shown in figure 1, together with a line representing the ideal curve (fitted using non-linear least squares for constant factor and offset). As is to be expected the measured results are slightly worse than the ideal with a speedup of 1.97 from 8 to 16 cores and a speedup of and 3.3 from 8 to 32 cores. We attribute this mainly to synchronization overheads. The physical machines for the c3 instances are based on two 16 core Intel E5-2680 CPUs according to amazon, which means that going from 16 to 32 cores inter-package communication will be required, which is significantly more expensive than intra-package communication. In addition the machines are also NUMA based with two affinity domains, which can also lead to higher access times for some memory accesses. In addition to synchronization overheads there are presumably some initialization and termination tasks that are not fully parallelized by GraphLab.

2.2 MULTI MACHINE SCALABILITY

As a next step we investigated scalability across multiple EC2 instances. We decided to separate this experiment from the previous single-machine scalability experiments, as we were expecting communication across machine boundaries to have significantly different properties, and therefore performance impacts, than communication within a machine. In contrast to the previous experiment this experiment uses a significantly larger seed set size to ensure that even with a larger number of machines the experiment would be long running enough to get significant results not dominated by initialization or termination cost.

This time we were using a variable number of c3.8xlarge instances, and again measuring the completion time. Besides offering the highest number of cores and thereby minimizing more expensive inter-machine communication, this instance type also offers the highest network speed (10Gbps Ethernet). To ensure fast communication we also ensured fast communication between our instances by launching them all with the same placement group which, according to amazon, should ensure full 10Gbps connectivity between the instances.

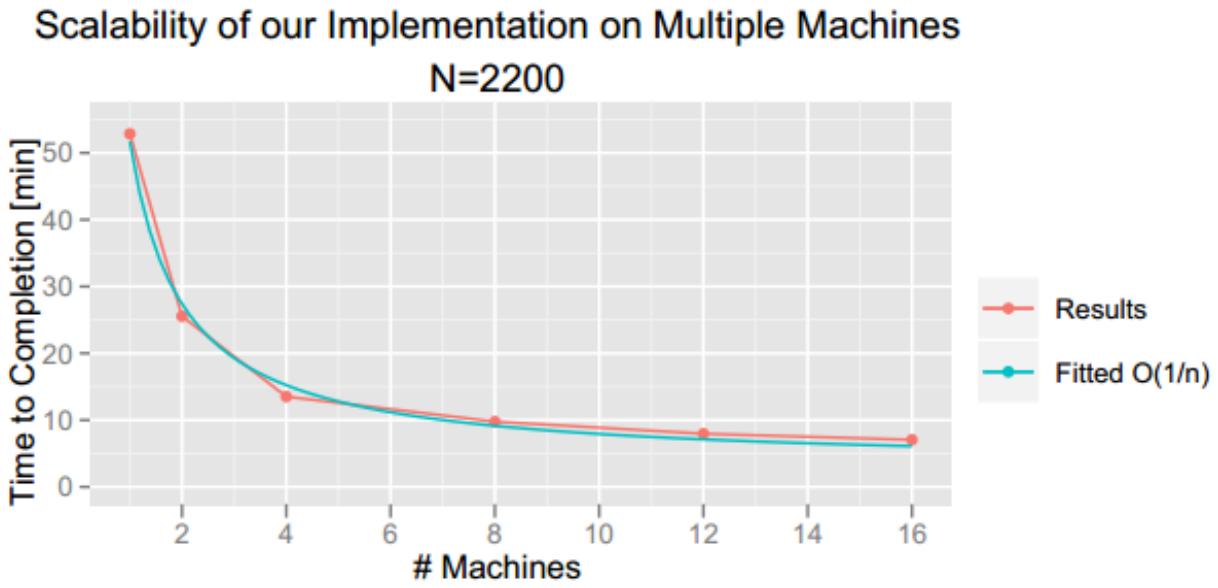

Figure 2. Scalability with increasing number of machines

Figure 2 shows the results for this benchmark, again overlaid with a fitted curve of the expected O(1/n). Again the measured values follow the expected trend. However this time the table of speed-ups below shows that the speedups flatten off much more pronounced for a larger number of machines. Until 4 machines the speedup is almost perfect, after that it drops significantly.

Machines	1	2	4	8	12	16
Speedup	1.0	2.1	3.9	5.4	6.6	7.5

Here our rationale for the reduced speedup is that more communication across machines will be required for synchronization and for communication between vertices on different machines. And as the number of machines increases it will be significantly more likely that vertices are handled by different machines, and therefore more likely that communication between two machines will be required for execution of the vertex programs (vertex signaling and gather).

2.3 COMPLETION TIME BY INPUT SIZE

Our final experiment was to investigate the relationship between the number of seed papers and the completion time. Looking at our algorithm we would expect a complexity of O(n) for phase 1 (calculating the distance of each node to all the seed nodes), and a complexity of $O(n^2)$ for phase 2, where we determine the LCA for each pair i.e. n^2 pairs, which leads to an overall complexity of $O(n^2)$. As an experimental setup we used 16 c3.8xlarge instances (512 cores in total).

The resulting measurements are shown in figure 3 together with a fitted curve for the expected complexity $O(n^2)$. As expected the completion time seems to be quadratic in the number of seed papers. The graph also shows that we can run our implementation for roughly 6700 SEED PAPERS IN 60 MINUTES.

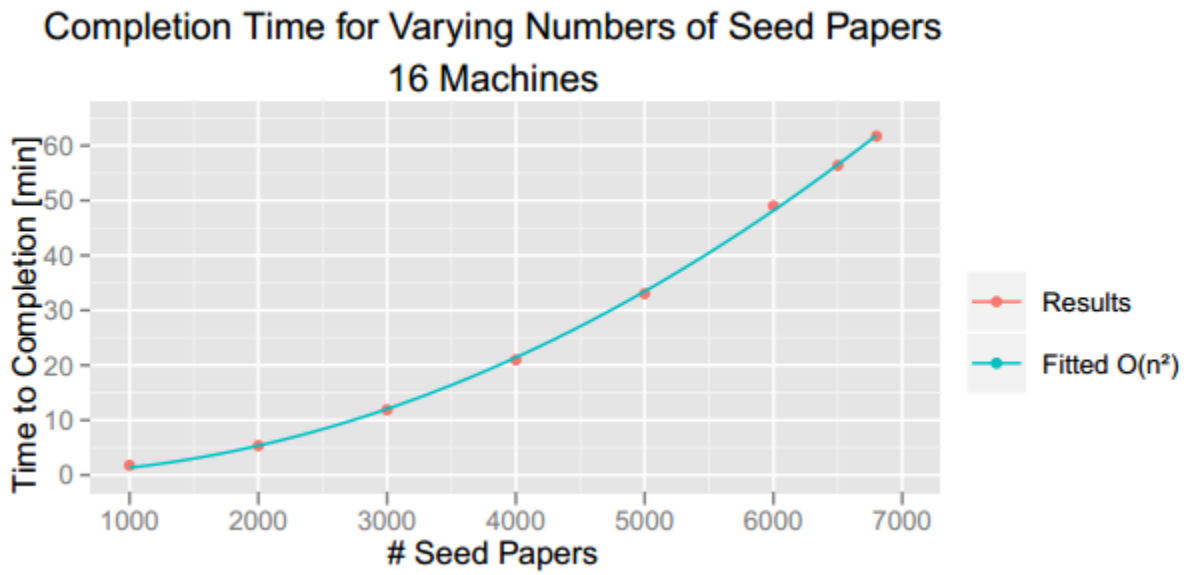

Figure 3. Relationship of input size to completion time

3. GRAPHLAB

We feel that graph lab was a great natural fit for our solution. The GraphLab abstractions we used, i.e. vertex program and fold over vertices, seem to provide ample opportunity for parallelism without manual intervention. We were impressed by how well our implementation scaled without specific tuning when we started evaluating it on multiple machines. GraphLab uses some of the following opportunities for parallelism in our implementation:

- Split input files, so the different splits can be read and parsed by separate cores/machines in parallel
- Vertex programs running on multiple cores in parallel, only communicating asynchronously
- Fold can be executed by cores and machines independently, and then combined

Even though GraphLab already provided strong performance results while relying on the basic features, we expect that using more advanced GraphLab features performance could be tweaked further.