Nolen Young, 11517296

Big Data, HW 3

10-27-19

* 1. Speedup refers to the change in performance as the number of processors change on a fixed workload. Scale up refers to how the performance changes as you scale the number of processes and the workload proportionately. Super-linear speedup is impossible in parallel algorithms because the speedup of an algorithm is bounded by network efficiency, for a super linear speedup each individual processor would have to work faster as you add more processes (impossible given that network taxes and startup costs are still present and increase as you increase the number of processors), and speed up is bound by the size in workload, at some point a given number of processes is too large for a given workload.
  2. Assume each instruction takes 1 time unit to execute. 2 = 50 time units, 4 = 43 time units, 8 = 42 time units. Speedup 2 = 1, speedup 4 = 1.16, speedup 8 = 1.19. An infinitely large machine will still take 40 time units to complete running, so 50/40 is 1.25.
  3. The three main groups of models are shared memory, message passing, and synchronized message passing models. Shared memory is good because it does not have any network penalties associated with the different processes accessing each other’s results, but two processes can read or write to the same address at the same time, which can cause memory corruption. Message passing is good because it allows memory to be distributed but comes at the drawback of network time cost. Message passing also allows completed tasks to continue working on other things while waiting for other processes to finish, allowing for more efficient run time. But messages can be out of sync, causing messages to be sent and never received. Synchronous message passing fixes the problem of messaging being out of sync but doesn’t allow the user to work on things while waiting for other processes to finish. This results in a minor slowdown compared to message passing at the benefit of stability.
     1. // key is the person
     2. // value is a list of friends
     3. Map (key, value) {
        1. Reducer value = value
        2. For each friend in value:
           1. Reducer key = buildSortedKey(person, friend)
           2. Emit(reducer key, reducer value);
     4. }
     5. Tuple buildSortedKey(person 1, person 2) {
        1. Returns sorted tuple of the two people
     6. }
     7. Main () {
        1. Get person;
        2. Get friends
        3. Map (person, friends);
     8. }
  4. Main () {
     1. MapReduce to get a list of all frequent keywords from all articles
     2. MapReduce to get a list of operate on result of last mapreduce to get a list of top ten used words
  5. }
     1. Map reduce to find all friends that are contained in the friend list of both I and j.
     2. Map reduce last result to find all friends that are two hops away that can reach both I and j.
  6. Q(G, u, v, d) {
     1. Connected nodes = Find all friends of u // ie all connected nodes
     2. For I < d
        1. Connected nodes = Map reduce connected nodes to find all nodes in the next hop
     3. If connected nodes is empty return false else return true