Dynamic load-balancing algorithm for a decentralized gpu-cpu cluster

Mert Değirmenci
Department of Computer Engineering,
Middle East Technical University
Related work

http://www2.computer.org/portal/web/csdl/doi/10.1109/ICPADS.2004.13161144

Rate-Based and Queue-Based Dynamic Load Balancing Algorithms in Distributed Systems:
Abstract
In this paper, we classify the dynamic, decentralized load balancing algorithms for heterogenous distributed computer systems into three policies ...

http://www.academypublisher.com/ijrte/vol01/no01/ijrte0101232236.pdf

An Algorithm for Dynamic Load Balancing in Distributed Systems with Multiple Nodes
Abstract
In a distributed network of computing hosts, the performance of the system can depend crucially on dividing up work effectively across the participating nodes. The reduction of the mean response time increases with the number of hosts,
Related work


Dynamic Load Balancing Algorithm using Execution Time Prediction on Cluster Systems

Abstract

The load unbalance is the major defect that reduces performance of a cluster system that uses parallel program in a form of SPMD (Single Program Multiple Data). The cluster system is a loosely-coupled distributed system, therefore, it has higher communication overhead than MPP. Dynamic load balancing can solve the load unbalance problem of cluster system and reduce its communication cost.


A Load Balancing Algorithm Using Prediction

From paper

Recently, Artificial Neural Network (ANN) has been proposed as a model for resolving a wide variety of problems in such diverse fields as combinatorial optimization computing, vision, and pattern recognition. Therefore, we adopt the ANN notion to resolve the task allocation problem as well load balancing problem posed herein.
Considerations

We need to design an algorithm using following facts:
- Spatial locality of patterns.
- Loosely coupled, heterogeneous, and possibly geographically dispersed network architecture.
Worker Algorithm

For a worker with n gpu, there are:
- n+3 cpu threads.
- 1 queue shared among all threads.
- 1 string holding current chunk supplier url.

Algorithm for n threads: thread index i

1. Set cuda device number to i, goto step 2.
2. If not done then goto step 3 else exit.
4. If queue is not empty then set front = queue.front, pop queue, goto step 5.
7. If pattern found then report this to supplier of chunk goto step 2 else goto step 2.
Worker Algorithm

Algorithm for socket listener thread:

1. Listen to socket, which was connected to master, goto step 2.
2. If new message arrived then handle message else goto step 1.

Algorithm for chunk bufferer thread:

1. Try lock mutex, goto step 2.
2. Set url to current chunk supplier url, goto step 3.
4. Set current chunk to retrieve one, append supplier url to chunk, goto step 5.
Worker Algorithm

Algorithm for selector thread:

1. For i = 0 to n, where n is the total number of chunk supplier urls, do step i, goto 2.
   i. Set array[i] = 0
2. For all chunk supplier urls do step a, goto step 3.
   a. Set start to current system time, goto step b.
   b. Send request for emergency variable of server, goto step c.
   c. If response arrived then goto step d else goto step c.
   d. Set end to current system time, set difference to end – start, goto step e.
   e. Set emergency to Response integer returned, goto step f.
   f. Set array[i] to EMERGENCY_CONS * emergency - PING_CONS * difference.
3. Set maxIndex to maximum of index of array[i], where i is 0 to n-1, goto step 4.
4. Lock mutex, goto step 5.
5. Set current chunk supplier url to chunkSupplierUrls[maxIndex], goto step 6.
7. Sleep for RELAXATION_CONS, goto step 1.
Chunk Supplier Algorithm

For each chunk supplier server, there are:
- One two dimensional array holding visited locations.
- One priority queue holding prioritized spatial locations.
- One integer holding total emergency variable.

Algorithm for reset.asp:

1. For index i is 0 to CHUNK_GRID_WIDTH do goto a, then goto 2.
   a. For index j is 0 to CHUNK_GRID_HEIGHT do goto i.
      i. Set visited[i][j] to “false”
2. While queue is not empty do goto x, then goto 3.
   x. Queue pop first element.
3. Set emergency variable to 0
Chunk Supplier Algorithm

Algorithm for nextChunk.asp:

1. If priority queue is empty then goto step 2 else goto step 3.
2. Return queue.front, pop first element of queue, exit.
3. Set first to first not visited location in grid, goto step 4.
4. Return chunk first, exit.

Algorithm for report.asp:

1. Set important to Request.patternContainingChunk, goto step 2.
2. For each chunk that is LOCALITY_THRESHOLD neighbourhood of important do a, go 3.
   a. Push the chunk to the priority queue.
3. For each coordinate in the database that is marked to be pattern Containing, do i.
   i. If found is lower than TEMPORAL_THRESHOLD then delete it and go continue.
   ii. Push the chunk to the priority queue with its found integer variable.
Advantages

Benefit of using temporal locality: Workers start working at previously pattern-detected regions.

Benefit of using spatial locality: Workers search locations with high probability of containing pattern.

Benefit of prioritizing: Locations with low number of occurrences are searched lastly.

Benefit of using bandwidth and transfer rate analysis: Workers work with closer chunk suppliers to themselves.
Disadvantages

- A log(n) overhead introduced to chunk supplier as a result of using priority queue.
- If number of prioritized chunks are not managed, there will also be memory problem.
- Extra protocol between Workers and Chunk Suppliers.
- O(n*n/(k+n)) overhead introduced to Worker for finding nearest Chunk supplier, where n is the number of chunk suppliers and k is the number of seconds that this thread sleeps.