International Research Journal of Engineering and Management Studies (IRJEMS)
Volume: 03 Issue: 03 | March -2019 e-ISSN: 2395-0126

Different Policies for Dynamic Load Balancing

Mohammad haroon
Research scholar
TMU mordabad

Dr (prof) Mohammad Husain
(AIET)
lucknow

Abstract

Many issues involve in dynamic load balancing such as how to measure the load of processing elements, how should load information we should collect and where the load is reside. How the real activity happening the different algorithms, these issue are usually grouped into several policies or component .we consider the dynamic load balancing algorithms consist of four component: a load measurement rule, and information exchange rule, load balancing operation, an initiation rule (define by location rule, selection rule, distribution rule) these issues are also group into a transfer policies, selection policies, location policies and information policies.

Key words: Load measurement, load information exchange rule load balancing operation.

Load measurement policies: measuring of load of every node is very important or the load balancing algorithms. Measuring the load in distributed system is very difficult task. Usually load is measure by metric these can be divided into two parts, Simple load indices and complex load indices

Simple load indices:

These consist the load on only single resource. These approaches only focus on the load on the CPU. Simple load indices include processor queue length over a given duration, the amount memory available, the context switch rate, the system call rate, CPU utilization.

Complex load indices: these consist a number of parameter, each relating to a single resource, such as CPU, disk, memory and network the load indices comprise the CPU load and the amount of free memory and number of concurrent user use the same node, paging rate, and the amount of ideal time at processing node. The load indices

should be easily to compute and correlate with the parameter e.g response time that is to be minimized. in a heterogeneous systems, the load indices from different node must be adjusted to make them comparable, for example if two different node have different processing power, their CPU utilization may have to divided by their processing power to compare their CPU utilization load indices value .

Load information exchange policies:

Information polices are responsible to collect all node information, it also decide what information is collected and when information is collected. the load information exchange policies can be broadly classified into three types.

Demand policies: Every node collects information when it needs and it to make a load sharing decision. A poll-limit is usually used. The main advantage is that load information is exchanged only when it is required.

Periodic policies*:* Information is distributed or collected at regular time intervals. This is simple to implement. However, it is important to determine the most appropriate distributions time period as overheads due to periodic communication Increase system load and reduce scalability. Here, a fixed amount of state collection overhead will be induced in the system because each node collects and maintains state-information of other nodes, regardless whether this information will be used. However, there is no polling delay when a task must be transferred.

State change policies. State-change policies generally have lower communication rates than periodic policies. However, if the state at a particular node does not change for a long period of time, the information held about that node will become stable. load-state information is unreliable, since there is no way of telling if the node has crashed or has just not sent a message due to a steady state. A newly joining node will not receive information concerning steady-state nodes, even if those nodes are suitable transfer partners. One way to improve the basic state-change policy is to introduce additional distributions messages, which are sent if the load-state does not change for a long period of time. These rules differ from demand-driven rules in that each node takes the initiative for distributing its own state instead of collecting other nodes information.

Where the load should be maintained?

Central approach: central approach can be used to calculate load-state information. This is collected from all of the nodes in the system and made available when a load-sharing decision must be made. Some centralized nodes are simply responsible for the collection and distributing of information, while others additionally act as matchmakers between sender and receiver nodes. Centralized node can work well in small or moderately sized systems, but can become communication bottlenecks when the system is scaled up. Where centralized components are used in the entire system, that system is vulnerable to the failure of the single component unless some form of backup or replication is provided, this increases complexity.

Distributed approaches: Distributed approaches are more difficult to build than their centralized counterparts. The semantics involved can be complex.

Each node collects information concerning the load state at other nodes in the system. Nodes autonomously base load sharing decisions on the information they hold. One advantage of distributed implementations is that the system is not vulnerable to the failure of any single node. There are also disadvantages there is no consistent system wide view of state, and each node holds different information depending on which other nodes it has communicated with, how recently that communication took place, and the delay experienced in that communication. This can lead to instability if there are significant differences in the views held.

How the load is collected?

First option collects the load from all systems. Another opinion is to divide the whole distributed systems into number of cluster and then collect the load from different cluster then finally calculates the over all load from all the clusters. The choices in between these two extremes use local load information collected from a certain domain of processing nodes in which size may be either fixed or variable. The global knowledge of the system's attributes (like the total work load) is prohibitive, due to the communication overhead produced. This is true for large-scale distributed systems. Thus, the technique of demanding global information is rejected, and partial information is used

instead, such as information of the neighborhood of a node.

Transfer policy:

A transfer policy determines whether a node is in a state to participate in a task transfer, either as a sender or a receiver. Many proposed transfer policies are threshold policies, which may be either based on fixed or adaptive thresholds. One way is to set one threshold value for the load imbalance (the difference between the largest and smallest loads on the nodes). If the detected load imbalance is bigger that a present threshold value, the transfer is initiated. An equivalent method to this is to set two threshold values, T1 and T2 by which the nodes are classified into three types, i.e., heavily loaded or sender (if loads higher than T1), lightly loaded or receiver (if loads lower than T2), and normally loaded otherwise . Depending on the algorithms, T1 and T2 may or may not have the same value. The choice of these thresholds is fundamental for the performance of the algorithm. Clearly, the best threshold values depend on the system load and the task transfer cost. At low loads and/or low transfer costs thresholds should favor task transfers, while at high loads and/or high transfer costs remote execution should be avoided. Although states that the optimal threshold is not very sensitive to system load, and present techniques that efficiently and in run-time adapt the threshold to the system load. Fixed threshold policies mean that the threshold values are not changed when system loads are changed. There are disadvantages with the fixed threshold policy. If the fixed threshold value is too small, this still causes “useless” job transfers. If the fixed threshold value is too large, the effect of using a load-balancing mechanism may be reduced. Other than using fixed threshold values, thresholds can be set in an adaptive (relative) fashion, by adjusting them when the global system load is changed. if the load of an individual node is above or below the average load over a certain domain (either the global or some local range) by a preset percentage, then load-balancing actions are initiated and load is balanced either locally or globally. In another adaptive approach to determining proper thresholds, the average load Lavg is determined first. Two constant multipliers, H and L, are used in computing the heavy threshold, T1, and light threshold, T2. H is greater

than one and L is less than one. These two values determine the flexibility and the effectiveness of a load-balancing mechanism. The heavy threshold, T1, is computed as the product of H and L average. Similarly, the light threshold T2 is computed as the product of L and L average.. The transfer policy may be either periodic or event-triggered. The algorithm may periodically check whether the node's state qualifies itself as a candidate for a task transfer. **Selection policy:** The role of selection policy is to select tasks for transfer. In sender-initiated schemes, busy nodes choose tasks to transfer to another node, whereas in receiver-initiated schemes, lightly loaded nodes inform potential senders of the types of task they are Willing to accept. The policy determines how much load, or how many tasks, to transfer. A task transfer may be

preemptive or non-preemptive. Preemptive transfers involve transferring a partially executed task. This is generally expensive, as it involves collecting all of the task's state. Non-preemptive task transfers involve only tasks that have not begun execution and hence do not require a transfer of the task state. A node may be overloaded and have no tasks available for no preemptive transfer if it is polled by a receiver. A selection policy should consider at least three factors. The overhead incurred in transferring the task should be minimized. No preemptive transfers and small tasks (small amounts of information) carry less overhead. The execution time of the transferred task should be sufficient to justify the cost of the transfer. Even if task execution is unknown, it should be possible to classify the tasks as short or long tasks, and to consider only the long tasks for migration. Some classification errors might be tolerated as load-balancing algorithms are quite robust with regard to this parameter. The number of location-dependent resources needed by the selected task should be minimal.

Location policy: The responsibility of location policy is to find a suitable transfer partner. Location policies can be distributed, each node selecting a transfer partner on the basis of locally held information. Location policy, corresponding to information policy, specifies the balancing domain for load-balancing actions; this could be global, nearest-neighbors, a group of random

polled nodes, or a set or cluster of nodes based on specified criteria. Alternatively, policies can be devised using a central information source. Busy nodes attempt to locate transfer partners that have low load levels in sender-initiated schemes. in receiver-initiated schemes, low-loaded nodes attempt to locate a busy node from which to transfer work. Five typical policies are listed below.

Existing load-balancing algorithms:

Two classes of well-known dynamic and distributed load-balancing algorithms are presented in this section. The focus is on the load-balancing algorithms utilizing partial information to make decision. Although some algorithms are initially presented for parallel computers, they are applicable in a distributed computing system with more or less deficiencies. Thus, these are also introduced here. Most load-balancing policies execute two activities that require communications: distribute its own load information and collect other nodes information and transfer tasks. If each node is required to interact with other nodes, it will have to use mechanisms such as broadcast, global gathering, long-distance communication which are not scalable and create intolerable overhead or congestion in systems with a large number of nodes. To reduce this overhead, in many policies, a node only exchange information and transfer tasks to its physical and/or logical neighbors'. These are usually called "neighbor-based" load-balancing algorithms. Clustering is another technique to tackle the problem. The nodes can be partitioned into clusters based on network transfer delay, where load-balancing operates on two-level: intra-cluster and inter-cluster via cluster managers or brokers. These are usually called "cluster-based" load-balancing algorithms. We will give corresponding discussion to these two classes of algorithms below.

Neighbors-based load-balancing algorithms: The neighbors'-based approach is a dynamic load-balancing technique that allows the nodes to communicate and transfer tasks with their neighbors' only. Each node balances the workload with its neighbors' so that the whole system will be balanced after a number of iterations. Since this technique does not require a global coordinator, it is inherently local, fault tolerant and scalable. Hence, this approach is a natural choice for load-

balancing in a highly dynamic environment. Among of the neighbor-based algorithms, we are interested in a couple of typical representatives, described as follows.

The gradient model: The gradient model (GM) is a demand driven approach. The basic concept is that under loaded nodes inform other nodes in the system of their state, and overloaded nodes respond by sending a portion of their load to the nearest lightly loaded node in the system. The resulting effect is a form of relaxation where tasks transferring through the system are guided by the proximity gradient and gravitate towards under loaded points. The scheme is based on two threshold parameters: the Low-Water-Mark (LWM) and the High-Water-Mark (HWM). A node's state is considered light if its load is below the LWM, heavy if above the HWM, and moderate otherwise. A node's proximity is defined as the shortest distance from itself to the nearest lightly loaded node in the system. All nodes are initialized with proximity of Wmax, a constant equal to the diameter of the system. The proximity of a node is set to zero if its state becomes light. A node's proximity may not exceed Wmax. A system is saturated, and does not require load-balancing if all nodes report proximity of Wmax. If the proximity of a node changes it must notify its near neighbors' gradient map of the proximities of under loaded nodes in the system serves to route tasks through the system in the direction of the nearest under loaded nodes. A task continues to transfer until it reaches an under loaded node or it reaches a node for which no neighboring nodes report a lower proximity.

Contracting within neighborhood: In thee contracting within Neighborhood method, two parameters need to be specified to make the contracting decision, minimum hops and maximum hop. Here, minimum hops specify the minimum number of hops needed for a drifting task to travel before it settles into the standing state. This parameter is used to ensure a newly created task will travel certain distances to reduce the horizon effect. The other, maximum hop, is the upper limit of travelling distance of a drifting task. It ensures that each newly created task will be sent only to a node within a fixed radius neighborhood from its source node. It prevents

unbounded message oscillations, and also induces locality which makes the communication between the creating and created tasks efficient. They keep track of the number of hops travelled so far for each task c, called c.hops. Thus, at each node, for a drifting task c, which is either created by themselves or received from other nodes, we have the following cases: if c.hops < minimum hops, a node will contract task c to its least loaded neighbor no matter its own load; if c.hops > maximum hop, task c will be retained locally, added to the local pool of messages, terminating its drifting state. Otherwise, the task will be contracted conditionally: if the load on the least-loaded neighbor is smaller than its own load, the task is contracted out to that neighbor. In this way, the newly generated task c travels along the steepest load gradient to a local minimum.

Summary: This paper has provided an extensive overview of existing load-balancing methods, with a focus on decentralized load-balancing approaches utilizing partial information to make decisions. As discussed, existing decentralised techniques, which rely on neighbours or clustering, are not applicable in a large-scale heterogeneous computational grid. The survey pointed out opportunities for improving the performance of decentralised load-balancing algorithms,

References:

1. Foster, C. Kesselman (Eds.), The Grid: blueprint for a new computing Infrastructure, Morgan-Kaufmann Publishers, 1st Edition 1999, 2nd Edition 2009.

2. Casanova, arnand legrund, and yves Robert, parallel algorithms, CRC press London

3. W. M. Jones, L. W. Pang, D. Stanzione, W. B. III. Ligon, Job communication Characterization and its impact on meta-scheduling co-allocated jobs in a minigrid,in: Proceedings of the 18th International Parallel and distributed Processing Symposium, 26-30 April 2008, pp:253-260.

4. Foster, C. Kesselman, S. Tuecke, The anatomy of the Grid: enabling scalable Virtual organizations, The International Journal of High Performance Computing Applications 15 (3) (2001) 200–222.