Journal of Computer Science 8 (1): 175-180, 2012
ISSN 1549-3636

Hybrid Algorithm for Optimal Load Sharing in Grid Computing

[1]D. Ramesh and [2]A. Krishnan
[1]Department of CSE, Anna University of Technology, Tiruchirappalli
[2]K.S. Rangasamy College of Technology, Tiruchengode

Abstract: Problem statement: Grid Computing is the fast growing industry, which shares the resources in the organization in an effective manner. Resource sharing requires more optimized algorithmic structure, otherwise the waiting time and response time are increased and the resource utilization is reduced. **Approach:** In order to avoid such reduction in the performances of the grid system, an optimal resource sharing algorithm is required. In recent days, many load sharing technique are proposed, which provides feasibility but there are many critical issues are still present in these algorithms. **Results:** In this study a hybrid algorithm for optimization of load sharing is proposed. The hybrid algorithm contains two components which are Hash Table (HT) and Distributed Hash Table (DHT). **Conclusion:** The results of the proposed study show that the hybrid algorithm will optimize the task than existing systems.

Key words: Hash Table (HT), Distributed Hash Table (DHT), High Performance Computing (HPC), distributed computing, information virtualization, double hashing reduces, grid environment, heterogeneous network, computing components

INTRODUCTION

Now a day, the scientific problem becomes very complex; therefore it requires more computing power and more storage space. These requirements are very common in an organization when dealing with current technological methodology. The past technologies such as distributed computing, parallel computing are not suitable for recent advancement. Because, the modern computer industry operating very large amounts of data which utilise more processing power and high storage volumes of data. Therefore, the Grid computing is proposed as effective resource management to the organization. The Grid computing is a growing technology, which is a High Performance Computing (HPC) branch for solving complex problems where two or more computing components of such as a traditional supercomputing centre, clusters of computers, a heterogeneous network, a distributed resources centre are integrated in a hardware and software infrastructure.

Grid computing has similar architecture to distributed computing but it is differentiated from almost all distributed computing paradigms by the following characteristic: the essence of grid computing lies in the efficient and optimal utilization of a wide range of heterogeneous, loosely coupled resources in an organization tied to sophisticated workload management capabilities or information virtualization.

As the Grid is growing in the modern era, it attracts researcher. There are variety of research activity is identified in the grid environment which is shown in the Fig. 1. In which the workflow scheduling and load balancing are the major research issue even till date. The architectural design of Grid is shown in the Fig. 2 and 3. AliEn RB (Boukerram and Azzou 2006, Suguna and Thanushkodi, 2011) is a Grid Broker which handles File transfer optimization, fault tolerance by multithreading and Push and pull task assignment. In Apples (Odeh *et al*., 2009; Latip *et al*., 2011), the Parameter study support, event-driven rescheduling, Centralized adaptive scheduling with heuristics and self-scheduled study queues are handled. In EZ-GRID Broker (Ibrahim and Salman, 2011; Richard *et al*., 2008), job handling, transparent file transfer, self-information service ith dynamic and historical data, Policy Engine Framework for provider policies are proposed.

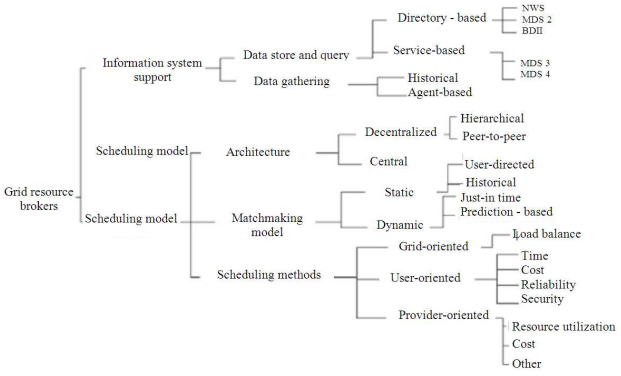

Fig. 1: Various research activity in the grid environment

Corresponding Author: D. Ramesh, Department of CSE, Anna University of Technology, Tiruchirappalli

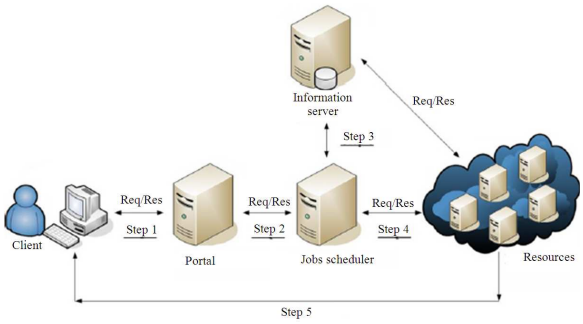

Fig. 2: Architectural design of job scheduler

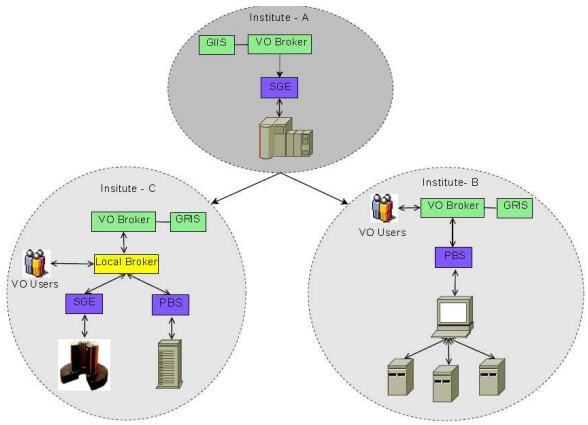

Fig. 3: Virtual organizaion based Grid Architectural design

In GRID BUS Grid Service system (Alshoaibi *et al.*, 2009), Failure management and application recovery, parameter study, API support, Economy-based and data aware scheduling are focused for solving. The GRUBER (Al-mazroi and Rashid, 2011) handles SLA-based resource sharing in multi-VO environment, disk quota considerations, internal site monitoring feature and various users oriented policies.

The APAC (Australian Partnership for Advanced Computing) Grid interconnects various Grid sites distributed across Australian Institutions and Universities. The APAC Grid uses a hierarchical information service, MDS-2. VPAC (Victorian Partnership for Advance Computing), which is a part of the APAC Grid, hosts the centralised GIIS (Grid Index Information Service:-a component MDS-2), while the remaining Grid sites run the GRIS (Grid Resource Information Service) that connects to the VPAC GIIS. A Grid resource broker who wishes to access the APAC Grid has to contact the VPAC GIIS, as contacting one of the other Grid sites running a GRIS would only allow access to information about that particular resource.

This isolation in resource information organisation in grids and among grids leads to the resource fragmentation problem. In this case, Grid users get access to only small pool of resources. Further, the institution hosting the root GIIS service is central point of contact for the overall system. Failure of the root GIIS can partition the system, and can lead to significant performance bottlenecks. To overcome the limitations of centralised and hierarchical information services, proposed a decentralised Grid resource information service based on a spatial publish/ subscribe index. It utilises a Distributed Hash Table (DHT) routing substrate for delegation of d-dimensional service messages. DHTs have been proven to be scalable, self-organising, robust and fault-tolerant. The proposed Grid resource discovery service organises data by maintaining a logical d-dimensional publish/subscribe index over a network of distributed Grid brokers/Grid sites. The spatial nature of the publish/subscribe index has the capability to respond to complex Grid resource queries such as range queries involving various attribute types including, those that have a spatial component.

Further, the resource discovery system is extended to provide the abstraction/facility of a P2P tuple space for realising a decentralised coordination network. The P2P tuple space can transparently support a decentralised coordination network for distributed brokering services. The P2P tuple space provides a global virtual shared space that can be concurrently and associatively accessed by all participants in the system and the access is independent of the actual physical or topological proximity of the tuples or hosts. The Grid peers maintaining the tuple space undertake activity related to job load-balancing across the Grid-Federation resources.

MATERIALS AND METHODS

The proposed method is a hybrid version of Hash Table (HT) and Distributed Hash Table (DHT). Therefore, this section further explains the methods and implementations of HT and DHT (Banejad *et al*., 2009; Latip *et al*., 2011; Salaheddin *et al*., 2009).

Hash Table (HT): The hash table is a fair and feasible way of resource optimization in the last few years. A hash table entry stores an item which is composed by a key and possibly some data, i.e., a pair of <k, d>. In hash table, every table position has a pointer, initially pointing to an empty value. When an item is inserted in the table, the pointer of the corresponding position refers to it. The hash table structure used by the minimal perfect hashing approach is designed such a way that all the items which are inserted directly in the table. This design is opt in many ways such that there are no empty entries in the hash table and no space is lost even when the

data volume in the item is large. The minimal perfect hashing avoids the use of memory space to keep the pointers and the space overhead does not depend on the items length. Some of the open addressing techniques are, linear hashing, quadratic hashing, double hashing, dense hashing and cuckoo hashing.

The linear hashing is the simplest open addressing schemes which uses a hash function and tests positions and so on, until it finds the term k being searched. Otherwise, if it finds an empty position, or if the sequential search reaches position h(k) after running over all other positions, the item being searched does not exist in the hash table. The main problem of linear hashing is, it degenerates in a sequential search when the number of terms n gets closer to the table size m, which causes a waste of time. Another issue is the waste of space caused by empty positions in the hash table.

The quadratic hashing is very similar to linear hashing that uses two additional parameters r and q. The parameter r indicates how many positions ahead from the current position of the next search for the term k will be performed and the parameter q indicates that the value of parameter r will be added to after each iteration. The quadratic hashing is expected to have a better performance when compared to linear hashing for higher load factors, since it prevents the production of clusters which delay the search for items. However, this method shares some problems found in linear hashing, e.g., the waste of space due to empty positions and the waste of time due to successive collisions when n gets closer to m. The quadratic hashing method may also have a smaller spatial locality when compared to linear hashing, as the pace r may become much larger than one. The period of search is defined as the number of entries that appear in a sequence from a particular initial position before an entry is encountered twice. The period of search should preferabaly be the same as the table size m or, at least, as large as possible. Otherwise, the table may appear to be full when there is still space available. If m is a prime number then the period of search for the quadratic hash method is m/2.

The double hashing also study in a way very similar to linear hashing, but instead of one function, it uses two: $h_1(k)$ and $h_2(k)$. The first one produces values in the range $(0, m^{-1})$, mapping the term into its position in the hash table, the same way the hash function in linear hashing does. The additional function $h_2(k)$ produces values in the range $1, m^{-1}$, which are used as steps in the process of finding empty positions. Values produced by $h_2(k)$ are relatively primes to the table size m. The double hashing reduces the problem of clustering in a better way than quadratic hashing does. This is because function $h_2(k)$ provides a different step d for each key k and the multiple step sizes produce a more uniform distribution of the used positions.

The Cuckoo hashing mimics the cuckoo chick's behavior and uses two hash functions $h_1(k)$ and $h_2(k)$ to get two possible table positions for a given term. When a term x has to be inserted in the structure, one of the two possible positions $h_1(x)$ or $h_2(x)$ is chosen. If the chosen position is already occupied, the term y contained there will be removed from the structure, yielding an empty position to the term x being inserted. Term y, in turn, has two possible positions given by $h_1(y)$ and $h_2(y)$. Consequently, y can be inserted differently from its prior position.

The hopscotch hashing is based on a mix of techniques from chaining, cuckoo hashing and linear hashing. This algorithm was designed for providing little synchronization overhead in a multi-processing environment and for high cache hit ratios. Furthermore, its performance does not degrade when the table load factor is high, i.e., more than 90%. In hopscotch hashing each key is mapped into an entry in an array of entries using a hash function h, but in case of collision it may be stored in one of the next H-1 neighboring entries, where H is a constant. This fixed range of entries is called a virtual bucket. Each entry keeps information on its keys' virtual bucket, with the purpose of finding the exact physical location of each key mapped into it. Each virtual bucket overlaps with other virtual buckets in the entry array.

The sparse hashing is based on a sparse array structure that uses little memory space and it is implemented as an array of groups A, where the number of groups in a sparse array of m entries is calculated as $G = \lceil m/M \rceil$. Each group stored in A[g], $0 \leq g < G$, is responsible for m indexes of the hash table, i.e., A [0] is responsible for the items in the range $[0, M^{-1}]$, A[1] for the items in the range $[M, 2\ M^{-1}]$ and so on. Each group g contains an array R_g that stores the actual items, an S-bit number to control the size of R_g and a bitmap B_g of size m. The bitmap B_g indicates the assigned indexes in the range $[0, M^{-1}]$. If $B_g[f]=1$, $0 \leq f < M$, then index f has a corresponding value stored in R_g. An item in group g with an offset f is not necessarily placed in position f of R_g, but in the position $R_g[j]$, where j is the number of bits set from $B_g[0]$ to $B_g[f^{-1}]$. Therefore, the array R_g is dynamically reallocated when new items are inserted into it. Thus, the size of R_g can differ among groups.

Although being very efficient in memory usage, sparse hashing is not designed to be efficient in time: each lookup needs to perform a sequential search through B_g to find the position of an item R_g. However, it presents a high spatial locality and this increases the number of cache hits when we perform insertions and lookups in the sparse hashing structure.

Distributed Hash Table (DHT): Now a day, Distributed Hash Tables (DHT) is a part of many Peer-To-Peer (P2P) applications in the Internet. To mention a few examples, DHTs are used to track the upload/download ratings in Bit Torrent and resolve host identifiers to IP addresses for Host Identity Protocol (HIP). Each DHT node maintains a routing table of its neighbors containing node Identifiers (IDs) and IP addresses. The main service provided by DHTs is routing a lookup query for a certain key to a DHT node that stores the value for that key.

Consider a DHT consisting of N nodes. Node IDs are assigned from an identifier space with a distance metric. Each node s maintains a routing table T_s of entries (u, IP_u), where u is a neighbor and IP_u is its IP address. Hence, s forwards messages to u via the underlying IP network. A message to a destination node d goes sequentially to nodes who's IDs are progressively closer to d according to the distance metric. If $v \in T_s$, then s is forward a message to v forming the one-hop path $s \rightarrow v$. Routing from s to d takes several hops forming a multi-hop path $s \rightarrow^{+} d$.

DHT is divided onto global and local parts. In the global part, a message is delivered close to the destination. In the local part, the destination is at a nearby node. The reasons for division are:

- Since a node is responsible for the keys closest to its ID, let a lookup message arrive to a node close to the key
- Various replication techniques support routing into an area of neighboring nodes
- A DHT node knows its neighborhood well, keeping close nodes to its routing table when possible. Obviously, global routing is more vulnerable to attacks

DHT routing is either iterative or recursive. With iterative routing each node on the lookup path returns the next-hop node v to the querying node. The latter then contacts v to get iteratively closer to the destination. With recursive routing, each node forwards lookups directly to the next hop nodes and the querying node receives a response from the destination. Iterative routing is more secure since a querying node can control the routing progress. Nevertheless, more network resources are consumed and iterative routing is not possible when a querying node cannot directly contact some nodes on the path, e.g., due to NATs. In this study, we consider recursive routing only.

Perfect Hash Function (PHF) and Minimal Perfect Hash Function (MPHF): A PHF is an injective function that maps keys from a set S to unique values. Since no collisions occur, each key can be retrieved from a hash table with a single probe. A MPHF is a PHF with the smallest possible range, that is, the hash table size is exactly the number of keys in S. MPHFs are widely used for memory efficient storage and fast retrieval of items from static sets. Differently from other hashing schemes, MPHFs completely avoid the problem of wasted space and wasted time to deal with collisions. Until recently, the amount of space to store an MPHF description for practical implementations found in the literature was O (logn) bits per key and therefore similar to the overhead of space of other hashing schemes. Recent results on MPHFs presented in the literature changed this scenario: an MPHF can now be described by approximately 2.6 bits per key.

RESULTS

A simple system design is shown in the Fig. 4. The distributed resource discovery system that supports multi-attribute based information search which handles multi-attribute lookups by creating a separate routing hub for every resource dimension. Each routing hub represents a logical collection of nodes in the system and is responsible for maintaining range values for a particular dimension while the notion of a circular overlay is similar to DHTs, it does not use any randomizing cryptographic hash functions for placing the nodes and data on the overlay. In contrast, this network is organised based on set of links. These links include the: i) successor and predecessor links within the local attribute hub; ii) k links to other nodes in the local attribute hub (intra-hub links); and iii) one link per hub (inter-hub link) that aids in communicating with other attribute hubs and resolving multi-attribute range queries.

In this model, the total routing table size at a node is k+2. When a node nk is presented with message to find a node that maintains a range value [li, ri], it chooses the neighbor ni such that the clockwise distance d(li, v) is minimized, in this case the node ni maintains the attribute range value [li, ri]. Key to message routing performance of Mercury is the choice of k intra-hub links. To set up each link i, a node draws a number $x \in I$ using the harmonic probability distribution function: pn(x) = 1 n log x. Following this, a node ni attempts to add the node n'in its routing table which manages the attribute range value r + (Ma −ma) ×x; where ma and Ma are the minimum and maximum values for attribute a.

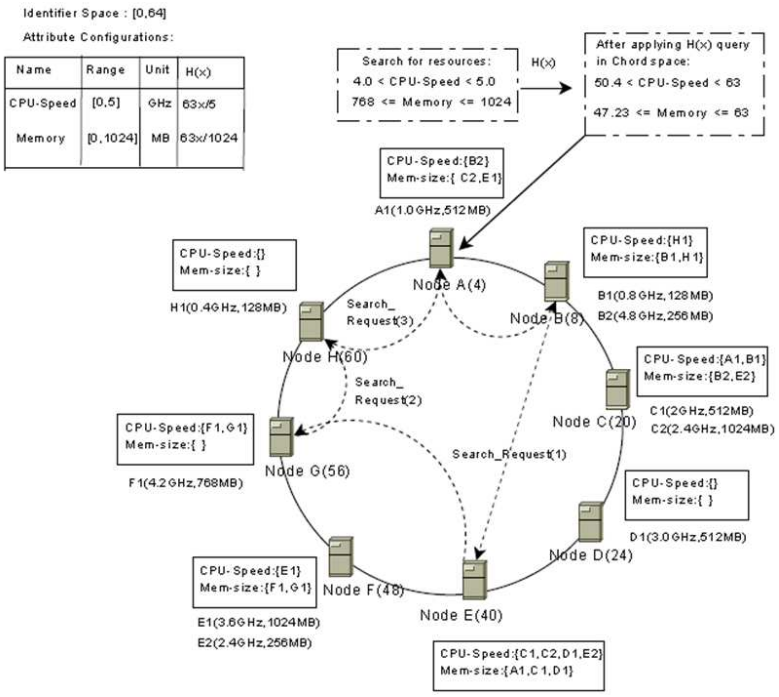

Fig. 4: Data flow on Hybrid Scheduling in Grid System

DISCUSSION

The objective of this study is to show that hybrid scheduling are, after the new recent results, a good option to index internal memory when static key sets are involved and both successful and unsuccessful searches are allowed. We have shown that our proposed hybrid scheduling provide the best tradeoff between space usage and lookup time when compared with other open addressing and chaining hash schemes such as linear hashing, quadratic hashing, double hashing, dense hashing, cuckoo hashing, sparse hashing, hopscotch hashing, chaining with move to front heuristic and exact fit.

CONCLUSION

We considered lookup time for successful and unsuccessful searches in two scenarios: (i) the MPHF description fits in the CPU cache and (ii) the MPHF description does not fit entirely in the CPU cache. Considering lookup time, the minimal perfect hashing outperforms the other hashing schemes in the two scenarios and, in the first scenario, the performance is better even when the compared methods leave more than 80% of the hash table entries free. Considering space overhead (the amount of used space other than the key-value pairs), the minimal perfect hashing is within a factor of O (logn) bits lower than the other hashing schemes for both scenarios.

REFERENCES

Al-mazroi, A.A. and N.A. Rashid, 2011. A fast hybrid algorithm for the exact string matching problem. Am. J. Eng. Applied Sci., 4: 102-107. DOI: 10.3844/ajeassp.2011.102.107.

Alshoaibi, A.M., A.K. Ariffin and M.N. ALmaghribi, 2009. Development of efficient finite element software of crack propagation simulation using adaptive mesh strategy. Am. J. Applied Sci., 6: 661-666. DOI: 10.3844/ajassp.2009.661.666.

Banejad, M., R.A. Hooshmand and M.T. Esfahani, 2009. Exponential-hyperbolic model for actual operating conditions of three phase arc furnaces. Am. J. Applied Sci., 6: 1539-1547. DOI: 10.3844/ajassp.2009.1539.1547.

Boukerram, A. and S.A.K. Azzou, 2006. Implementation of load balancing algorithm in a grid computing. Am. J. Applied Sci., 3: 1810-1813. DOI: 10.3844/ajassp.2006.1810.1813.

Ibrahim, L.F. and H.A. Salman, 2011. Using hyper clustering algorithms in mobile network planning. Am. J. Applied Sci., 8: 1004-1013. DOI: 10.3844/ajassp.2011.1004.1013.

Latip, R., H. Ibrahim and F.A. Al-Hanandeh, 2011. Scientific data sharing using clustered-based data sharing in grid environment. Am. J. Econ. Bus. Admin., 3: 146-149.

Latip, R., H. Ibrahim and F.A. Al-Hanandeh, 2011. Scientific data sharing using clustered-based data sharing in grid environment. Am. J. Econ. Bus. Admin., 3: 146-149. DOI: 10.3844/ajebasp.2011.146.149.

Odeh, S., R. Faqeh, L.A. Eid and N. Shamasneh, 2009. Vision-based obstacle avoidance of mobile robot using quantized spatial model. Am. J. Eng. Applied Sci., 2: 611-619. DOI: 10.3844/ajeassp.2009.611.619.

R. J.A. Richard, Ajay A. Joshi and C. Eswaran, 2008. Implementation of computational grid services in enterprise grid environments. Am. J. Applied Sci., 5: 1442-1447. DOI: 10.3844/ajassp.2008.1442.1447.

Salaheddin Odeh, Rasha Faqeh, Laila A. Eid and Nihal Shamasneh, 2009. Vision-based obstacle avoidance of mobile robot using quantized spatial model. Am. J. Eng. Applied Sci., 2: 611-619. DOI: 10.3844/ajeassp.2009.611.619

Suguna, N. and K.G. Thanushkodi, 2011. An independent rough set approach hybrid with artificial bee colony algorithm for dimensionality reduction. Am. J. Applied Sci., 8: 261-266. DOI: 10.3844/ajassp.2011.261.266.