A Hierarchical Distributed Approach for Mining Molecular Fragments

Christoph Sieb[1], Giuseppe Di Fatta[2], and Michael R. Berthold[1]

[1] ALTANA Chair for Bioinformatics and Information Mining
Department of Computer and Information Science, University of Konstanz
Box M 712, 78457 Konstanz, Germany
{sieb,berthold}@inf.uni-konstanz.de
[2] School of Systems Engineering, University of Reading
Reading RG6 6AY, United Kingdom
difatta@reading.ac.uk

Abstract. Recently, two approaches have been introduced that distribute the molecular fragment mining problem. The first approach applies a master/worker topology, the second approach, a completely distributed peer-to-peer system, solves the scalability problem due to the bottleneck at the master node. However, in many real world scenarios the participating computing nodes cannot communicate directly due to administrative policies such as security restrictions. Thus, potential computing power is not accessible to accelerate the mining run. To solve this shortcoming, this work introduces a hierarchical topology of computing resources, which distributes the management over several levels and adapts to the natural structure of those multi-domain architectures. The most important aspect is the load balancing scheme, which has been designed and optimized for the hierarchical structure. The approach allows dynamic aggregation of heterogenous computing resources and is applied to wide area network scenarios.

1 Introduction

Due to the enormous amount of data created by many of today's transactional applications, it has become necessary to parallelize the corresponding mining algorithms to attain reasonable response times. One of these applications in the field of drug discovery is related to the *High Throughput Screening* (HTS) technology. The HTS process is widely used to identify potential compound candidates for further research in the drug discovery process. HTS is able to screen more than 100,000 compounds a day for several activities, such as inhibition of HIV or cancer cells. The screened compounds are recorded in a transaction-like database together with their activity level. As the number of candidate compounds is extremely large, it is useful to extract features from the active compounds and use them to reduce the number of relevant candidates.

Key features are those molecular fragments that occur frequently in active compounds but infrequently in non-active ones, and therefore represent promising starting points. These discriminative fragments can be extracted by modeling

the compounds as undirected labeled graphs and applying *Frequent Subgraph Mining* (FSM) algorithms [9] to them. Even though the available FSM algorithms use sophisticated methods to speed up the mining process, the scalability issue can only be solved by increasing the computational power of the underlying system. Increasing the power of a single processor machine is limited by current technology and physical laws. One possible approach to overcome these limitations is to partition the original problem into smaller subtasks and allocate them to several processors.

Large companies and institutions typically deploy many ordinary, heterogeneous desktop PCs and servers at different locations with several security policies. They are often underutilized and therefore offer a large-scale computing resource pool. The challenge is to make this pool accessible for computing-intensive applications to solve problems without the need of expensive special hardware.

The first work on distributed mining of frequent molecular fragments [5] describes a centralized master/worker approach, which partitions the induced depth-first search tree to distribute the mining task. Very often such search problems exhibit a highly irregular tree shaped computation, and, in some cases as in molecular fragment mining, the complexity of subtasks cannot be estimated. In this case it is essential to adopt a dynamic load balancing policy. Typically, for FSM algorithms, the search space representation in memory is much bigger than the database size and thus, parallel approaches distribute the induced search tree instead of the database. The second approach, designed as a completely distributed peer-to-peer system [6], solved the inherently scalability problem due to the bottleneck of the central master. Both approaches require direct channel communication to exchange messages. However, in many real world applications the potential computing nodes are not always directly accessible from each other due to security restrictions. In those environments both approaches cannot exploit the potential computing power and therefore represent an *architectural bottleneck*.

In this work we present the first hierarchical distributed system for FSM and other highly skewed search tree problems applied in the context of ordinary computer technology. The hierarchical system aligns to the inherent hierarchy of those multi-domain networks to overcome the described drawback of a centralized or peer-to-peer system. The load balancing scheme takes into account the specific challenges of highly skewed search tree problems and the hierarchical structure respectively. Furthermore, it maintains locality to keep the system scalable. The tests show that the system performs similar to the centralized approach but additionally enables access to multi-domain clusters restricted by real world security policies.

The rest of this paper is structured as follows. The next section discusses related approaches to hierarchical distributed systems and FSM. In section 3, we briefly describe a concrete sequential FSM algorithm on which our distributed approach is tested. In section 4, we introduce the centralized master/worker approach, followed by section 5, which presents the architecture of the hierarchical

system and describes the hierarchical load balancing scheme. Section 6 describes the experiments we conducted to verify the performance of the hierarchical approach. Finally, we provide concluding remarks.

2 Related work

Many dynamic load balancing (DLB) algorithms for irregular problems have been proposed in literature and their properties have been studied [7]. In the field of hierarchical distributed computing, Antonis et al. describe in [2] a hierarchical load balancing scheme that needs to know the number of participating nodes in advance and builds up a logical binary tree.

In [4] Dandamudi and Lo present a load sharing policy for identical nodes by arranging them in a hierarchical structure. This work addresses hierarchical distributed systems and, in particular, sender and receiver initiated load balancing techniques. In [8] a hierarchical load balancing scheme is described that works close to the operating system. The scheme is implemented on a massive parallel system where the processors are connected by high speed networks. The described load balancing scheme, however, needs to know the current load level, which is not known in search tree problems with highly skewed data.

Finally in [10], a whole framework is proposed to provide a common platform for distributed applications. In this framework, it is necessary to know the problem size of a task to enable good load distribution. Unfortunately, this is impossible for most data mining problems as the size of the underlying search space is unknown a priori.

3 Molecular fragment mining

The problem of selecting discriminative molecular fragments in a set of molecules can be formulated in terms of frequent subgraph mining in a set of graphs. Molecules are represented by attributed graphs, in which each vertex represents an atom and each edge a bond between atoms. Each vertex carries attributes that indicate the atom type and a possible charge and each edge carries an attribute that indicates the bond type. Frequent molecular fragments are subgraphs that have a certain minimum support in a given set of graphs, i.e., are part of at least a certain percentage of the molecules. Discriminative molecular fragments are contrast substructures that are frequent in a predefined subset of molecules and infrequent in the complement of this subset. In this case, two parameters are required: a minimum support ($minSupp$) for the focus subset and a maximum support ($maxSupp$) for the complement.

The algorithm organizes the space of all possible fragments in an efficient search tree. An example of such a search tree is depicted in Figure 1. The algorithm is based on an exhaustive depth-first search strategy. Each node of the search tree represents a candidate frequent fragment. A search tree node evaluation comprises the generation of all the embeddings of the fragment in the molecules. An embedding of a fragment consists of references into a molecule

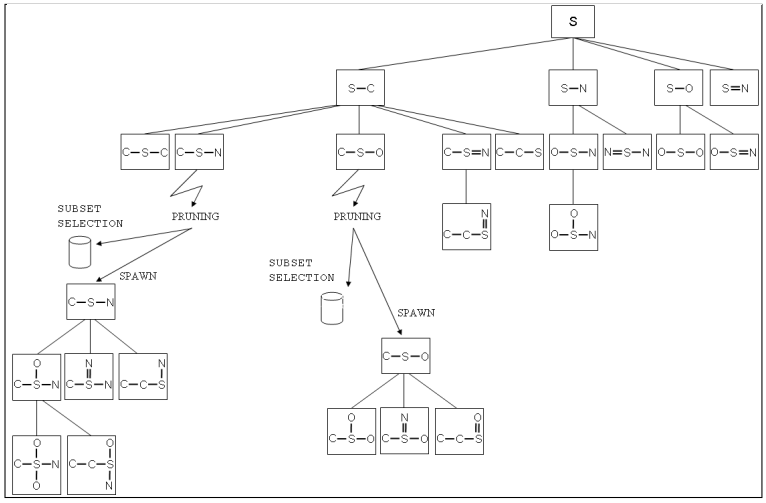

Fig. 1. Search tree partitioning

that point out the atoms and bonds that form the substructure. The embedding list allows both a fast computation of the fragment support in the active and inactive molecules and a fast extension to bigger fragments. When a fragment meets the minimum support criterion, it is extended by one bond to generate new search tree nodes. When the fragment meets both criteria of minimum support in active molecules and maximum support in the inactive molecules, it is then reported as a discriminative frequent fragment.

The search starts from a root node with a single atom and is iterated for each frequent atom type. The algorithm prunes the DFS tree according to three criteria. The support-based pruning exploits the anti-monotone property of fragment support. The size-based pruning exploits the anti-monotone property of fragment size. And, finally, a partial structural pruning is based on a local order of atoms and bonds. For further details of the algorithm we refer to [3].

After introducing the underlying application algorithm, the next section describes the centralized master/worker approach to introduce the basic DLB scheme.

4 The master/workers approach

There are two aspects in parallel and distributed systems that influence scalability, overall performance and flexibility. The first aspect is the logical communication topology, while the second aspect concerns the load balancing scheme. The logical communication topology is important with respect to scalability issues and the flexibility to adapt to administrative policies, e.g. secure subnets.

Nevertheless, load balancing must be tightly incorporated and must respect the logical structure.

In the master/workers approach, the topology is a star scheme, where the worker nodes perform the actual mining task and the master is responsible for distributing the workload equally. The distribution of the mining task is done by partitioning the search space, i.e. each worker explores a different part of the search space and the master node merges partial results. Search space partitioning is performed by pruning a branch of the depth-first search tree induced by the mining algorithm. To this aim, an external description of the pruned search node has to be generated and donated to an idle worker (see Figure 1). This description must be appropriate to set up the same state of the mining process for the receiving worker to explore the search subtree. A donated search node describes a molecular fragment that occurs in a subset of the molecular database. A pruned fragment with the information necessary to restart the computation in any of the participating nodes is subsequently referred to as a job. Each computing node has its own replica of the dataset. In order to decrease the computation overhead of re-embedding the fragment in the molecules, the ID list of the supporting molecules can be included in the description (subset selection).

The adopted DLB scheme is a receiver-initiated policy because data mining applications are typically computation bounded problems. The master node manages a pool of unprocessed jobs from which it serves child requests.

The master node sends job pruning requests to worker nodes (called donators) once the job pool size J is below a given threshold. This threshold is derived from the number of worker nodes C and a relative threshold value α_L called the relative job pool threshold. Thus, a soft state request is periodically sent while the following boolean expression holds $J < J_L$, where $J_L = \alpha_L \cdot C$ is the absolute job pool threshold. If the number of jobs is below the threshold J_L the master node sends j job requests to its children, whereas j is calculated by the difference of the job pool size J and the absolute pool threshold J_L plus the number of pending requests c: $j = J_L - J + c$

The master node chooses a worker with a previously assigned job that can donate a part by pruning its search tree. In dynamic load balancing schemes the idea is to ask the node with most work. Unfortunately, in this kind of search problems the actual complexity of a subtask is not known in advance. The load can only be estimated by the heuristic that older assignments represent bigger jobs. To decrease the probability of a bad selection and to avoid too many requests to a single node the requester selects the donator by a Ranked Random Polling [6].

When a worker has received a job request, it applies two heuristics which increase the probability of pruning a big job, referred to as *pruning heuristics*. The first one applies a focus support threshold $suppTH$, which is greater than the minimum focus support $minSupp$. Fragments with larger focus support $supp_F$ have a higher probability of being further extended instead of being

discarded due to the downward closure property. Therefore, only fragments for which $supp_F \geq suppTH$ holds are considered for pruning.

The second heuristic exploits the *local order* defined in the sequential algorithm [3] to identify larger jobs. This order defines which atoms of the current fragment can be extended. A fragment with many extendible atoms creates a larger sub tree in the search space. Thus, fragments can only be pruned if the number of extendible atoms over the total number of atoms is larger than a threshold t_L, $0 \leq t_L \leq 1$.

Whenever a job is finished at a worker node, the result is reported to the master node, which merges the partial results. Once the master's job pool is empty and no assignments are left in the assignment list, the search is over.

5 The hierarchical distributed system

The advantages of the centralized approach are an easy-to-handle topology, direct communication paths, and a global state maintained by the master, which offers very good load balancing capabilities. However, in many real world scenarios the master often is not able to access each node directly. Even, in cases where nodes are accessible, long communication delays from the master to several workers would reduce the performance.

We adopt a hierarchical communication topology based on a tree. Tree hierarchies correspond to the actual structure of multi-domain systems, in which just one node in a domain is accessible from nodes of other domains.

In the following subsections, we describe the management of the logical topology and outline the hierarchical load balancing scheme.

5.1 Topology management

The hierarchy is structured into administration nodes (admins) and worker nodes (workers). The root node represents a special admin.

Worker nodes perform the actual mining task, i.e. they explore a part of the search tree. Furthermore, worker nodes create new subtasks for idle workers by pruning a node of their current search tree. Once the subtask has finished, the result is sent to the parent admin node. Workers are the leaf elements in the communication tree.

Admin nodes manage other nodes and merge the partial results received from their children. Admins represent inner nodes organized in a tree with one or more hierarchical levels. All admins (except root) eventually propagate their aggregated partial results to their parent node.

All computing nodes, except the root, join the system by registering themselves to their parent node. Even if there is no restriction on the hierarchy depth, flat hierarchies have to be preferred in order to avoid long communication paths. However, the branching factor at admin nodes should be limited to avoid a bottleneck similar to the one of the centralized master-workers approach. It should be noticed that, in contrast to the centralized system, the hierarchical topology

can be extended as soon as an admin may represent a bottleneck. A dynamic topology management is out of the scope of this paper.

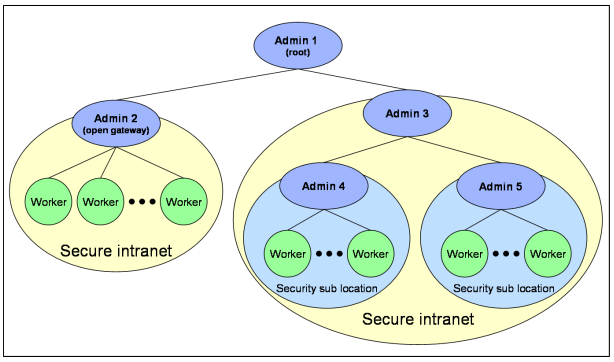

Fig. 2. Schematic example hierarchy

Figure 2 shows an example hierarchy with a root node managing two admins. Admin 2 manages several workers within a secure intranet, which are only accessible from the admin (gateway). Admin 3 is also part of a secure intranet but manages two further security sub-locations.

The next section describes the DLB scheme, which has been designed to work efficiently in hierarchical topologies.

5.2 The load balancing scheme

As in the centralized master/workers approach, the mining task is distributed by partitioning the search space. In this case, the management task is also distributed over several admins at different levels of the hierarchy. Each admin node manages a job pool to serve idle nodes in its subtree. When the job pool size is below the threshold, the admin sends job requests to its child nodes to solicit the generation of new subtasks (search tree pruning). The admin is also logically connected to the rest of the system through its parent node. For a global load distribution the admin has to send job requests to its parent as well, which in turn will forward the request to another branch of the communication topology. However, this naive approach would involve the whole system into the job acquisition process making vain the scalability potential of the hierarchical topology. For the sake of scalability it is necessary to preserve locality in subtrees of the hierarchy. In the next sections we introduce the concept of local and global load balancing that ensure locality in the subtrees.

Local load balancing An admin node performs Local Load Balancing (LLB) to distribute and balance the load among its children (either admins or workers).

From the local point of view, the admin still performs a centralized DLB (section 4). An admin node manages a pool of unprocessed jobs and a threshold J_L is used to trigger job-donation requests to its children. We refer to J_L as the absolute *local* job pool threshold. Children are treated the same way regardless of being workers at a leaf level or further admins. As in the centralized approach, when a worker receives a job-request, it will prune a part of the local search tree according to the *pruning heuristics* (see section 4) and send the new job to the parent node. In case a job request is sent to an admin node, it is forwarded by applying the same LLB policy at each intermediate admin until a worker is reached.

Global load balancing While LLB tries to balance the load among children , upward requests to a parent node generate load balancing activities referred to as Global Load Balancing (GLB). An upward request is triggered by the job pool size according to a relative *global* threshold α_G, when $J < J_G = \alpha_G \cdot C$.

Locality is a crucial aspect for the scalability of hierarchical systems. Communication and load balancing activities within subtrees have to be autonomous up to a certain degree to prevent that global communication may limit the scalability. An admin node must first try to satisfy demands for new jobs within its own subtree and, as last extent, it will send a request to its parent node. Therefore, the global threshold is set below the local one ($\alpha_G < \alpha_L$), avoiding global requests when they are not necessary.

The thresholds J_L and J_G are triggering requests for new jobs. The complementary aspect in DLB is the donation of jobs once a node receives a request. Donations to child nodes are granted immediately. In contrast, donations to the parent node should only be granted if the job pool has enough jobs. This is because the job pool of the donating node could fall below the global threshold J_G which would result in an immediate request to the parent which just received the donated job. To avoid this problem and to strengthen the locality aspect, a third threshold parameter is introduced called the donation threshold α_D, which is related to upward job donation. Thus, donations to the parent node are only granted if $J >= \alpha_D \cdot C = J_D$.

Figure 3 illustrates this threshold in combination with J_L and J_G. Admin 2 is below the local and global threshold and thus performs requests to its children and also to its parent node. Admin 1 is below the local threshold but still above the global one. Therefore, it performs downward requests to the child nodes but no upwards requests to its parent. In the case that Admin 1 receives a request from its parent node it would not donate a job from the pool as it retains below the donation threshold J_D. The global threshold and the donation threshold are complementary, i.e. a computing node sending a request to the parent node only receive jobs from subtrees that have enough jobs. In general, donating jobs upwards involves more communication than downward donation. To avoid small jobs resulting in frequent GLB, a computing node donating upwards chooses the biggest job from the current job pool (according to the *pruning heuristics*).

This strategy reduces global communication and, thus, contributes to system scalability.

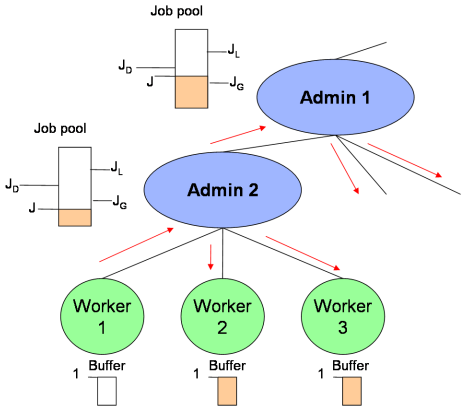

Fig. 3. Hierarchical topology

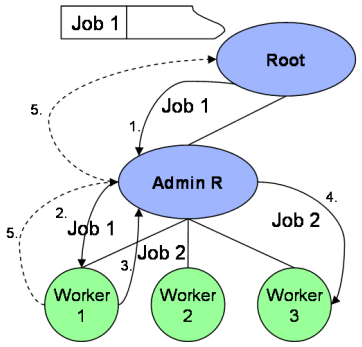

Fig. 4. Pruning Problem

The previous two sections described the hierarchical DLB scheme. Another important aspect in hierarchical systems is the aggregation of information and states. Aggregation makes it possible to approximate a global state for good DLB performance and keep the system scalable anyway. The next section describes this in more detail.

Hierarchical aggregation In centralized systems decisions can be made with a global state ensuring easy and effective load balancing. Hierarchical systems must aggregate system information as it is not possible to maintain the global state in all levels of the hierarchy, i.e. a node should only know about its children and its parent node without requiring knowledge about the structure in deeper levels or other branches of the tree.

There exist two important problems, which have to be solved. The first one concerns the assignment lists and the second one concerns the termination detection of the hierarchical distributed mining run.

As described in section 4 the efficiency of the donor selection policy depends on the correctness of the assignment list. However, in the hierarchical system this cannot be guaranteed across more than one level in the hierarchy as Figure 4 shows.

In step 1 Root assigns job 1 to its child Admin R and adds the job to its assignment list. Admin R assigns the job to Worker 1, which starts a mining process. The worker prunes a part from job 1 resulting in a new job 2. If job 1 completes before job 2, job 1 is reported as finished to Root (via Admin R) even though a part of the original job is still mined in the subtree. As job 1 will be

removed from the assignment list of Root the branch is no longer considered for pruning.

Thus, it is not enough for an admin to remember the job ID and its assignment time. The reason is that job pruning can also occur at deeper levels and thus is not recognized by the admin.

To solve this problem, we adopted a job identifier (job ID) with a hierarchical structure which allows aggregated state information about jobs in a subtree. When a job is propagated downwards the logical topology, at each intermediate node (admin) a digit is appended in a dotted notation (e.g., "3.5.2"). The hierarchy of job IDs respect the hierarchies of tasks in the search tree. In particular, the prefix identifies the parent task and the last appended digit is a sequential counter that uniquely identifies the subtask. The admin node is responsible for maintaining, incrementing and appending the last digit to each job assignment. Job completion messages from children are aggregated by means of their ID prefix to detect the completion of the parent task. When this last condition is met, the admin node sends the aggregated job completion message up to its parent.

For each prefix an admin node maintains a *completion entry*, containing the following fields.

- A: Number of downward assignments for the given prefix
- B: Number of completed assignments for this prefix
- C: Number of jobs in the pool

Each job assigned downwards is counted in its completion entry of the corresponding prefix. In case a new job is created by pruning a search tree node, the job is given the same prefix and is assigned to the parent. The parent increments C of the entry and puts the job into the job pool. If the job is assigned upwards, C is decremented and the job is not considered in this subtree any more. A job is only reported as completed if the job pool (C) for a given prefix is empty and the number of downward assignments (A) and the number of completed assignments (B) is equal. In this case no part of the original job exists in the subtree.

This system ensures that the assignment lists always represent the correct state of jobs in a subtree, thus enabling the donator selection policy to work properly in the whole hierarchy.

The termination detection problem can also be solved by the hierarchial ID system. As described above, the system ensures that jobs are only reported as completed if all descendant jobs inside the same subtree have also been reported as completed. The initial job, created by the root node, represents the whole mining task. When the root detects the completion criterion (see above) for the initial job, the mining run is over. The reason is, that all jobs created during the whole mining run descend from the initial job and the subtree of the root node represents the whole tree.

6 Experimental results

To show the runtime behavior and the functionality of the hierarchical load balancing scheme we set up different long-distance hierarchical system config-

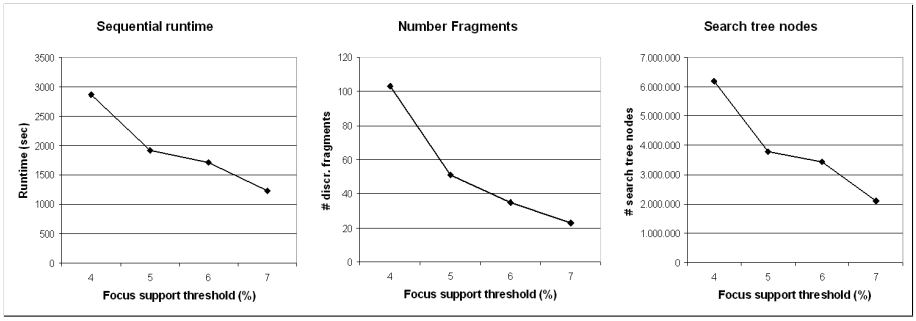

Fig. 5. Sequential performance measures

urations. A pool of 57 heterogenous computing nodes (1.5GHz / 0.5GB up to 3.4MHz / 1GB) were located at the ICAR-CNR[3] in Italy (24 nodes) and at the University of Konstanz (UniKN) in Germany (33 nodes).

The test configurations are based on two basic scenarios. The first scenario has two worker clusters at different locations (UniKN, ICAR) each managed by an admin, which also resides at the corresponding location. The root node is located at UniKN and manages the admins of both clusters. The second scenario introduces a third worker cluster located at UniKN.

In the hierarchical tests the number of participating workers varies from 12 to 48. The workers are distributed equally among the clusters. Table 1 shows the different configurations of both scenarios.

The hierarchical configurations are compared to the centralized approach. This involves the same computing nodes, which however, are directly managed by the master node.

All tests have been conducted on the well-known and freely available *NCI AIDS Antiviral Screen* screening dataset from the National Cancer Institute (NCI) [1]. A total number of 37171 molecules have been divided into a focus and a complement partition according to an activity threshold (0.5), respectively of 325 compounds and 36846 compounds.

For speedup analysis the sequential algorithm was executed on each of the heterogenous machines with focus support thresholds (see Section 3) of 4%, 5%, 6% and 7%. The runtime of the fastest machine is applied for later speedup analysis. Figure 5 shows the sequential runtime as well as the number of reported discriminative fragments and the number of search tree nodes that have been explored. The graphs depict the exponential problem space. Even though the number of reported fragments is quite low due to the complement threshold filtering, the number of search tree nodes that have been explored shows a huge search space.

[3] Istituto di Calcolo e Reti ad Alte Prestazioni, Sezione di Palermo, Consiglio Nazionale delle Ricerche, Italy

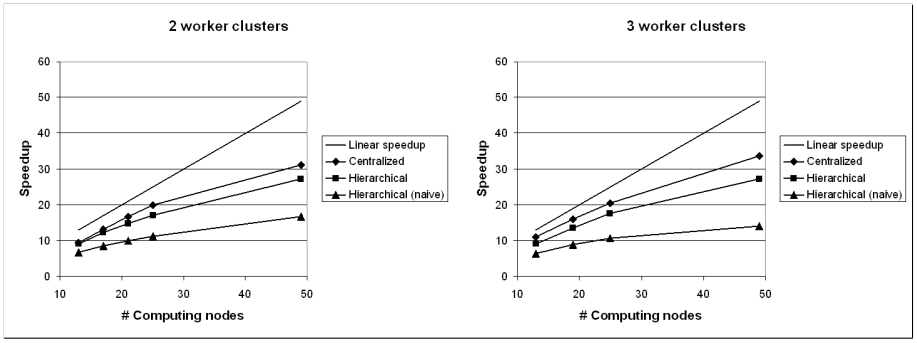

Fig. 6. Speedup comparison

In the distributed tests, the focus support threshold is fixed to a relative value of 4%. For the given focus partition of 325 molecules a fragment must occur in at least 13 of them to be frequent. The complement threshold is set to 0.01%. Therefore, a frequent fragment is also a discriminative fragment if it occurs in a maximum of 4 molecules of the complement partition.

The two graphs of Figure 6 show the speedup values of the hierarchical tests (see Table 6). For each configuration the speedup values are compared to the speedup of the corresponding centralized approach and also to the values of the naive hierarchical approach without the differentiated hierarchical load balancing (thresholds and aggregation).

As expected, the speedup of the hierarchical system is below the one of the centralized system. This is due to the global state maintained by the centralized approach, which provides better DLB quality. Nevertheless, the performance of the hierarchical approach is still very good. In both scenarios (2 and 3 clusters) the naive approach performs poorly due to the missing job-state aggregation (bad donor selection) and the unrestricted parent requests (no DLB thresholds differentiation), which results in unnecessary job prunings. As a result the idle times increase. The graph in Figure 7 shows the average idle time for the naive and proposed hierarchical DLB approaches.

The average idle time increases when number of clusters increases, i.e. with a more distributed system. As expected, the naive version shows longer idle times. Note the bigger difference between the idle time in the 2 and 3 clusters configuration of the naive version with respect to the proposed one. If the hierarchy becomes more complex, the impact of insufficient aggregated state information and undifferentiated load policies becomes more relevant. The experimental results have provided evidence of the effectiveness of the proposed hierarchical DLB scheme applied to multi-domain environments.

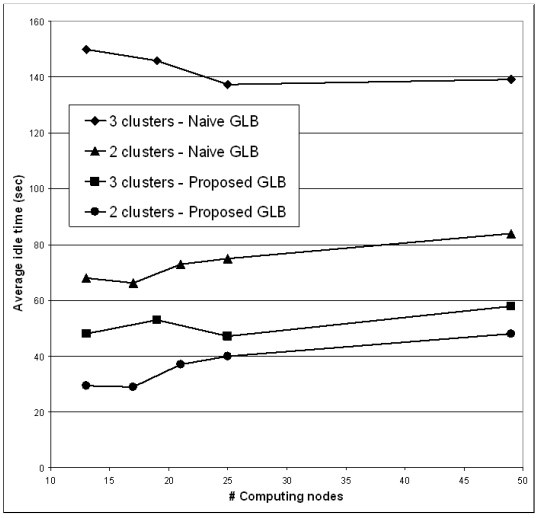

Fig. 7. Average idle time

Basic scenario	Worker UniKN	Worker ICAR	All nodes
2 clusters	5	5	13
	7	7	17
	9	9	21
	11	11	25
	23	23	49
3 clusters	2x3	3	13
	2x5	5	19
	2x7	7	25
	2x15	15	49

Table 1. Test configurations

7 Conclusions

In this paper we have presented a distributed approach to the frequent subgraph mining problem for computational environments that are characterized by a hierarchical communication topology. The system widens the architectural bottleneck of centralized and peer-to-peer systems, which cannot operate in multi-domain environments due to security restrictions. The proposed approach was successfully applied to the discriminative molecular fragments mining problem, but can also be applied to many problems based on a search tree in which the search space is unknown in advance.

The system applies a sophisticated hierarchical aggregation as well as a differentiated DLB scheme to reduce the drawback of missing global state information (as opposed to centralized approaches). The system maintains locality within the hierarchical structure to keep it scalable.

The experimental results show that the hierarchical approach performs close to the centralized one and is able to exploit potential computing power not eventually accessible.

Future work will deal with a topology management that dynamically and efficiently organizes participating nodes by taking into account both security policies and network delays.

8 Acknowledgements

This work was partially supported by the DFG Research Training Group GK-1042 "Explorative Analysis and Visualization of large Information Spaces". We

also thank Christian Stolze of the University of Konstanz, Germany and Pietro Storniolo of ICAR-CNR, Italy for their administrative support of the computing facilities.

References

1. http://dtp.nci.nih.gov/docs/aids/aids_data.html.
2. K. Antonis, J. Garofalakis, I. Mourtos, and P. Spirakis. A hierarchical adaptive distributed algorithm for load balancing. *Journal of Parallel and Distributed Computing*, 64:151–162, 2004.
3. C. Borgelt and M. R. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. IEEE International Conference on Data Mining (ICDM 2002, Maebashi, Japan). pages 51–58, December 09-12, 2002.
4. S. Dandamudi and K. Lo. A Hierarchical Load Sharing Policy for Distributed Systems. *5th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, Haifa, ISRAEL*, 1997.
5. G. Di Fatta and M. R. Berthold. Distributed mining of molecular fragments. *Proc. of IEEE DMGrid, Workshop on Data Mining and Grid of IEEE ICDM*, 2004.
6. G. Di Fatta and M. R. Berthold. Dynamic load balancing for the distributed mining of molecular structures. *IEEE Transactions on Parallel and Distributed Systems, Special Issue on High Performance Computational Biology*, 17(8), August 2006.
7. V. Kumar, A. Grama, and V. N. Rao. Scalable load balancing techniques for parallel computer. *Journal of Parallel and Distributed Computing*, 22(1):60–79, July 1994.
8. R. Pollak. A hierarchical load balancing environment for parallel and distributed supercomputer, International Symposium on Parallel and Distributed Supercomputing, Fukuoka, Japan. 1995.
9. T. Washio and H. Motoda. State of the art of graph-based data mining. *ACM SIGKDD Explorations Newsletter*, 5(1):59–68, July 2003.
10. Y. Xu, T. K. Ralphs, L. Ladanyi, and M. J. Saltzman. Alps: A framework for implementing parallel search algorithms, The Proceedings of the Ninth INFORMS Computing Society Conference. 2004.