Available online at www.sciencedirect.com

ScienceDirect

Microprocessors and Microsystems 31 (2007) 273–281

MICROPROCESSORS AND MICROSYSTEMS

www.elsevier.com/locate/micpro

Decentralized load balancing for highly irregular search problems

Giuseppe Di Fatta [a,*], Michael R. Berthold [b]

[a] School of Systems Engineering, The University of Reading, Whiteknights, Reading, Berkshire, RG6 6AY, UK
[b] Department of Computer and Information Science, University of Konstanz, Box M712, 78457 Konstanz, Germany

Available online 30 January 2007

Abstract

In this paper, we present a distributed computing framework for problems characterized by a highly irregular search tree, whereby no reliable workload prediction is available. The framework is based on a peer-to-peer computing environment and dynamic load balancing. The system allows for dynamic resource aggregation, does not depend on any specific meta-computing middleware and is suitable for large-scale, multi-domain, heterogeneous environments, such as computational Grids. Dynamic load balancing policies based on global statistics are known to provide optimal load balancing performance, while randomized techniques provide high scalability. The proposed method combines both advantages and adopts distributed job-pools and a randomized polling technique. The framework has been successfully adopted in a parallel search algorithm for subgraph mining and evaluated on a molecular compounds dataset. The parallel application has shown good scalability and close-to linear speedup in a distributed network of workstations.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Distributed computing; Peer-to-peer-computing; Dynamic load balancing; Irregular search problems; Subgraph mining

1. Introduction

A large class of applications relies on implicitly defined search trees and are based on a Depth First Search (DFS) strategy, e.g. problems solved using the divide and conquer strategy. In order to reduce running times and to cope with larger problem instances, partitioning a DFS tree, i.e. parallel backtracking [1], has been widely and successfully adopted in many applications. It is quite straightforward to partition the search tree to generate new independent jobs, which can be assigned to idle processors for asynchronous execution. In general, a static partition of the search space can be effectively adopted when job running times can be estimated. However, in many real-world problems the search space is highly irregular and no work load estimation is available. In the present work, we define the irregularity of a computational problem in terms of the empirical probability distribution of the task complexity, given a partitioning strategy. In particular, highly irregular problems are characterized by a power-law distribution of the task running times in a wide range. For such irregular problems it is essential to provide a dynamic partitioning strategy along with a Dynamic Load Balancing (DLB) policy.

Many DLB algorithms for irregular problems have been proposed in the literature and their properties have been studied. However, most of them rely on assumptions that do not hold in general. Some DLB techniques for irregular problems are based either on the assumption of uniform or bounded task times, or on the availability of workload estimates. In [2], uniform time tasks are assumed. In [3], it is assumed that the smallest task time is comparable to or greater than network communication time for a task. In [4], the computation, is first evenly partitioned among processors and, subsequentially task migration is adopted to maintain load balance in the system. In [5], several DLB algorithms are analyzed and their scalability properties are provided in terms of the isoefficiency analysis. This study particularly addresses irregular problems where it is not possible to estimate the size of work at processors. Nevertheless, the assumptions in [5] require the guarantee that the computation cost of jobs is always greater than

* Corresponding author.
E-mail addresses: G.DiFatta@reading.ac.uk (G. Di Fatta), Michael.Berthold@uni-konstanz.de (M.R. Berthold).

the relative transmitting time. However, this is not the case in highly irregular problems where a poor work-splitting mechanism generates a high number of trivial tasks and makes the DLB policy inefficient. Moreover, such a problem becomes more relevant in large computational environments, like Grids, with multi-domain and heterogeneous resources.

We present a distributed computing framework for highly irregular search problems, whereby no reliable workload prediction is available. The framework is based on a Peer-to-Peer (P2P) computing environment and a DLB policy. The DLB method employs a randomized technique that is enhanced by global statistics to cope with highly irregular search problems. The adoption of heuristics based on global statistics provides good load balancing performance, while the P2P approach and the randomized technique provide high scalability.

The proposed distributed computing framework has been developed in Java according to a multi-agent model. It has been successfully adopted to deploy and test a parallel application of the subgraph mining problem for the discovery of relevant fragments in molecular compounds.

The rest of the paper is structured as follows. In the next section we introduce the test application, i.e. the frequent subgraph mining problem applied to molecular compounds, and discuss the irregular computation that characterizes it. In Section 3, we introduce the general architecture of the distributed computing framework and describe the dynamic load balancing policy. Section 4 presents a performance evaluation of the parallel application. Finally, we provide conclusive remarks and future research directions.

2. Frequent subgraph mining

Algorithms to find frequent subgraphs in a set of graphs have received increasing attention over the past years. E.g., in molecular biology it is often desirable to find common topological properties in large numbers of drug candidates. Existing methods attempt to implicitly organize the space of all possible subgraphs in a lattice, which models subgraph relationships. The search then reduces to traversing this lattice and reporting all subgraphs that fulfill the desired criteria.

The Frequent Subgraph Mining (FSM) problem [6] is formulated in analogy to the Association Rule Mining (ARM) problem [7]. While in ARM the main structure of the data is a list of items (itemset) and the basic operation is the subset test, FSM relies on graph and subgraph isomorphism.

The subgraph isomorphism test is known to be an NP-complete problem [8]. Furthermore, there exists no known polynomial algorithm for isomorphism testing of general graphs, although the problem has not been shown to be NP-complete. In [9] the problem has been assigned to a special graph isomorphism complete complexity class, which falls between the P and NP-complete classes. However, it is known that it can be solved in polynomial time for many restricted classes of graphs, such as bounded-degree graphs [10].

Nevertheless, the combinatorial nature of the problem poses a great challenge. The computational complexity of the underlying problem and the large search space to be explored often render sequential algorithms useless. In such cases, distributed and parallel high performance computing becomes necessary for practical applications.

2.1. Molecular fragments discovery

The problem of selecting relevant 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 (i.e., the chemical element), a possible charge, and whether it is part of an aromatic ring. Each edge carries an attribute that indicates the bond type (single, double, triple, or aromatic). 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. We assume that the molecular compounds in the dataset can be classified in two groups. We refer to the two classes of molecules as the focus set (active molecules) and its complement (non-active molecules). For example, during the high throughput screening of drug candidates, compounds are tested for a certain active behavior and a score associated to their activity level is determined. In this case, a threshold (*thres*) on the activity value allows the classification of the molecules in the two groups.

The aim of the data mining process is to provide a list of molecular fragments that are frequent in the active dataset and infrequent in the inactive dataset. These topological fragments carry important information and may be representative of those components in the compounds that are responsible for a positive behavior. In this case, two parameters are required: a minimum support (*minSupp*) for the focus subset and a maximum support (*maxSupp*) for the complement.

Such discriminative fragments can be used to predict activity in other compounds and to guide the synthesis of new ones. For example, they can be adopted as features in a multidimensional space for molecular compounds classification [11] and clustering [12].

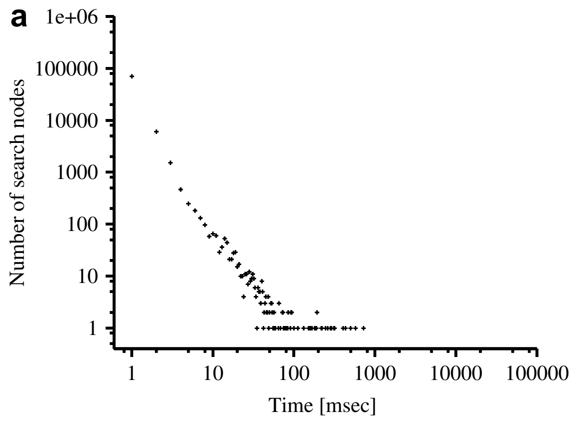
Based on existing ARM algorithms for market basket analysis [7,13] proposed methods for subgraph mining conduct depth-first [14,15] or breadth-first searches [16].

2.2. Irregular search space

An analysis of the sequential algorithm for molecular fragments discovery in [14] points out the irregular nature of the search tree. An irregular problem is characterized by a highly dynamic or unpredictable domain. In this application, the complexity and the exploration time of the search tree, and even of each single search tree node cannot be estimated. The data mining nature of the problem makes the time required to visit a node unpredictable. In our tests a single node exploration can take from few milliseconds to several minutes. It is known that the pruning techniques, which this kind of search algorithms heavily rely on, make the estimation of the tree exploration time very difficult. Depth and fan of the search tree are also unpredictable.

In order to provide evidence of the above considerations, we collected statistics of the running time required by the sequential algorithm for the expansion of each search tree node (Fig. 1(a)) and for the complete visit of the associated subtree (Fig. 1(b)). Both are characterized by a power-law distribution. This may not come as a surprise with regard to the subtree visiting time, since it is well known that this kind of distribution is typical for aggregations of multi-scale hierarchies such as trees. However, it is interesting that the node expansion time shows the same type of distribution. This can be tentatively explained by the structure of the input data and the node expansion step in the sequential algorithm. Each search tree node represents a molecular fragment and the expansion step consists of the extension of the fragment in all possible successors by adding a bond and, eventually, an atom. Intuitively, this produces a high number of fast fragment extensions due to the extension rules of the algorithm [14], and a small number of computationally long extensions. The irregularity of the search space implies that the work load of a task is not known in advance and cannot be estimated. As a consequence, in a parallel approach a static load balancing policy cannot be adopted or it would produce very poor results. Moreover, no assumption can be made on the lower bound of subtask workloads. We cannot assume, for example, that the subtask transmission time is less than the computation cost associated. This makes most dynamic load balancing policies unsuitable for this problem and others with similar characteristics.

In the next section, we describe the proposed distributed computing framework for problems with a highly irregular search space, which has been designed for heterogeneous and distributed computing resources.



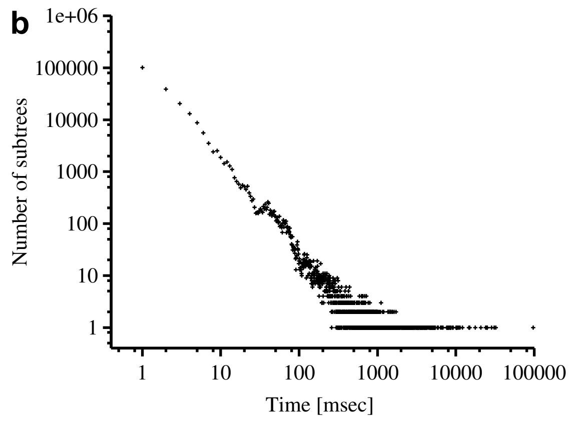


Fig. 1. Running time distribution in the search tree (*minSupp* = 10% and *maxSupp* = 1%). (a) Search-node expansion time. (b) Search subtree visiting time.

3. Distributed computing framework

The parallel application is mainly a work load distribution process based on dynamic partitioning of the search space, a dynamic load balancing policy and a peer-to-peer computing model. A P2P communication framework naturally fits with a receiver-initiated DLB approach and provides good scalability properties. Each peer registers at a centralized bootstrap node to join the system and to contribute to the computation task. Peers can directly exchange system information and computation subtasks, since each process implements both client and server functionality. The bootstrap node provides initial configuration information and a dynamic peer directory service.

The model of each peer process is a multi-agent system (Fig. 2), where each agent is an independent local entity (Java thread) with a specific role. Agents exchange and react to asynchronous messages by taking proper actions. Three main agent protocols have been designed for, respectively, the peer-to-peer system management, the DLB policy and the job statistics maintenance. MPI-like methods have also been adopted for synchronous communication.

The *Communication Manager* (CM) provides an interface to network communication and is responsible for the peer-to-peer system management (the setup and maintenance of the distributed environment). The implementation of our P2P computing framework allows the dynamic

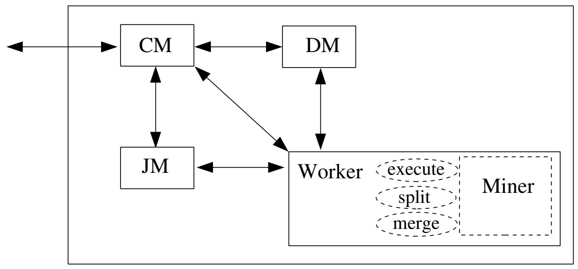


Fig. 2. Peer architecture.

joining of peers and provides a basic fault-tolerance mechanism in case of abrupt peer disconnection. CM also implements some typical MPI methods, which are used to implement higher level functionalities, such as agent message delivery, synchronization barriers, reduction operations, etc.

The *Data Manager* (DM) provides an interface to access the input data in a distributed environment. Typically, the DM at the boot node reads the data from a local file and transfers them to the other DMs. DM also provides basic mechanisms to partition the data among the peers. In our tests the entire dataset is simply sent to each peer. In the present work we tested a parallel application that adopts a task parallel approach and does not include any form of data parallelism.

The *Job Manager* (JM) implements the DLB algorithm based on distributed job pool, as described more in details in the Section 3.2.

The *Worker* executes assigned jobs by means of the algorithm (*Miner*) for a particular application. It provides an interface to which a sequential algorithm must comply in order to be embedded in the system. Three methods have to be implemented to adapt the sequential code. The *execute* method is used to invoke the execution of a sequential task. This simply corresponds to a wrapper of the sequential algorithm to comply with the *Worker* interface. A partitioning method for work splitting (*split*) divides a sequential task into two independent subtasks. The work splitting mechanism depends on the particular data mining applications. In applications based on a search tree, the search nodes are typically stored in a stack and the work splitting mechanism corresponds to the selection of one or more elements of the local stack to generate and donate a subtask to an idle Worker. A third method (*merge*) combines two partial results. In the particular data mining application, this operation involves graph and subgraph isomorphism tests to discard duplicates and to select only the closed frequent fragments. The three methods will be invoked asynchronously at each peer of the distributed system according to the parallel algorithm and the dynamic load balancing policy.

The *Worker* must also provide mechanisms to convert internal/external representations of jobs (e.g. search tree nodes). Since we adopt a distributed memory model, internal representations of the algorithm data have to be converted to external ones, which do not depend on the particular memory address space. This generates extra computation that contributes to the overall parallel overhead. The internal representation depends on the particular application and the sequential algorithm [14]. e.g. In the tested application, for the external representation we have enhanced the SMILES syntax for molecular compounds in order to embed partial state information from which the computation can be restarted in a different machine.

The parallel algorithm (Fig. 3) is composed by three phases. In the first phase the P2P computing environment is configured for the execution of the computational tasks. The second phase is the exploration of the search space and corresponds to the computation in the sequential algorithm. When the search is completed, in the third phase, local lists of partial results have to be merged and collected at a single node to be reported to the user.

The search space partitioning, the DLB policy, the termination detection of the search phase and the final reduction of results are discussed in the next sections.

3.1. Search space partitioning

Partitioning a DFS tree, i.e. parallel backtracking [1], has been widely and successfully adopted in many parallel applications. In general, it is quite straightforward to partition the search tree to generate new independent jobs, which can be assigned to idle processors. In this case, no synchronization is required among remote jobs.

A job assignment contains the description of a node removed from the search tree of the donor *Worker*, which becomes the root node from which the receiving *Worker* starts a new search. The job assignment must contain all the information needed to continue the search from exactly the same point in the search space.

Often the efficiency of search algorithms relies on advanced pruning techniques, which require specific state information to be propagated in the nodes along the search tree. Thus, a job description must include the search-node state necessary to rebuild the same conditions in the distributed search tree as in the sequential one (e.g. structural pruning state in [14]). This requires an explicit representation of the relevant implicit state for the donated search nodes.

During the parallel execution of subtasks, each *Worker* maintains only a local and partial list of results that are found.

The partitioning of the search space is carried out by the *Worker* upon request of the local JM, according to the DLB algorithm.

3.2. Dynamic load balancing

In general, a DLB policy has to provide a mechanism to fairly distribute the load among the processors using a small number of generated subtasks to reduce the communication cost and the computational overhead. In particular, in applications based on irregular search-trees the DLB policy has to carefully select (1) a suitable subtask donor among all the workers and (2) a non-trivial subtask among the search nodes in the local stack of the donor. The quality of both the selection of donors and the generation of new subtasks is fundamental for an effective and efficient computational load distribution. These two tasks are carried out, respectively, by the DLB policy and the work splitting-mechanism.

In problems with uniform or bounded subtask times the generation of either too small or too big jobs is not an issue. In our case, irregular job granularity may decrease

***Bootstrap* node:**	**Each *Peer* node:**
Setup phase	
▪ setup and manage the P2P system ▪ distribute input data	▪ register to bootstrap node ▪ request configuration and data
Search phase	
▪ create and assign first job (search tree search node) ▪ maintain global job statistics	▪ execute assigned jobs ▪ request and donate jobs according to the DLB policy ▪ partition the local search tree for donation ▪ send requests to potential donors when idle ▪ maintain and propagate a local list of potential donors ▪ maintain and propagate local job statistics for termination detection
Reduction phase	
▪ Perform a reduction operation as root of a logical tree topology	▪ Perform a reduction operation on a logical tree topology

Fig. 3. Pseudocode of the parallel application.

the efficiency and limit the maximum speedup tremendously. While a coarse job granularity may induce load imbalance and a bound on the maximum speedup, a fine granularity may decrease the efficiency of the distributed system and more processing nodes would be required to reach the maximum speedup. Thus, it is important to provide an adaptive mechanism to find a good trade-off between load balancing and job granularity.

In order to accomplish this aim we introduce a mechanism at the donor to reduce the probability of generating trivial tasks and of inducing idle periods at the donor processor itself. A set of heuristic rules [17] filters out many potentially trivial nodes. Although the number of very small subtasks has been significantly reduced, they cannot be completely discarded by the use of the rules; the visiting time of search tree nodes still has a power-law distribution. A randomized DLB technique further restricts the possible choices for new subtasks by selecting promising donors. This, once more, changes the distribution of the subtask visiting time. The final actual distribution is not available because it is the result of a parallel execution where subtasks are dynamically generated; task donation alters the task (subtree) at the donor itself.

Both components, the work splitting technique for the subtask generation and the DLB policy for donor selection, contribute to the overall DLB efficiency, scalability and to its suitability to irregular problems and to heterogeneous computing resources. The DLB approach we have adopted, the *Ranked Random Polling* (RRP), is a receiver-initiated algorithm based on a decentralized job-pool system and a centralized server for job statistics, as described in the next sections.

3.2.1. Decentralized job pool system

Each JM manages a local pool of available jobs, which can be assigned to remote JMs upon request and to the local *Worker* when idle. This simple communication protocol is shown in Fig. 4. Requests are based on a soft state. Each JM also keeps a list of donated and not completed jobs in order to support mechanisms for fault tolerance and termination detection. The JM also issues requests to the local *Worker* for the partitioning of the local search space (job generation). These activities are regulated by two thresholds of the job pool. A job can be donated to a remote JM only if the current job pool size is above a donation threshold J_d, which is constant and typically set to 1 (since there is only one local *Worker*). This way at the completion of a local job, the request and reception of a new one can be overlapped to the execution of a job from the local pool. Overlapping computation with communication of job request/assignment helps to avoid, or at least to reduce, most of the communication overheads. Only the latency of the first job assignment and of the last job-completed message cannot be avoided at all.

The job generation activity is enabled when the job pool size is below a spawning threshold J_s. In a centralized job

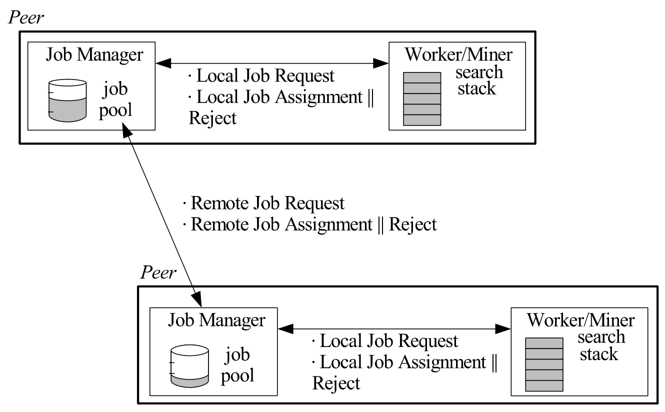


Fig. 4. Decentralized job-pool system.

pool approach, J_s can be easily set to the number of participating nodes. All nodes send job requests to the centralized job pool manager. In a decentralized system the threshold J_s must be independently set and adapted at each peer.

The determination and dynamic adaptation of the threshold J_s is critical for the effectiveness of the DLB in the system. J_s adaptation depends on the donor selection policy as described in the next section.

3.2.2. Donor selection

When the job pool size is below the threshold J_d, the JM selects a donor among the other peers to request a new subtask. A job request sent to a peer has the effect that the remote JM will solicit its local *Worker* to remove a search tree node from the local stack in order to distribute part of the search tree. In general, not all workers are equally suitable as donor. Workers that are running a mining task for a longer time, have to be preferred. This choice can be motivated by two reasons. The longest running jobs are likely to be among the most complex ones. And this probability increases over time. Secondly, a long job-execution time may also depend on the heterogeneity of the processing nodes and their loads. With such a choice we provide support to the nodes that are likely overloaded either by their current mining task assignment or by other unrelated processes.

Each JM keeps a local directory of potential donors and performs a random polling over them to get a new task. The probability of selecting a donor from the list is not uniform. In particular, we adopt a simple linearly decreasing probability, where the donor list is ordered according to the starting time of their current job. This way, long running jobs have a high probability of being further partitioned, while most recently assigned tasks do not.

Similarly, if a peer has a high rank in its own list of donors, it can expect to receive many remote job requests. As a consequence, it will increase or decrease the local threshold J_s accordingly.

The effectiveness of the donor selection and of the J_s threshold regulation depends on the correctness of the local statistics. In order to collect and propagate statistics of job executions, we adopted two mechanisms based on, respectively, a centralized statistics server (typically at the bootstrap node) and a decentralized P2P information distribution. The centralized server provides initial configuration information and a dynamic peer directory service. At the starting and at the completion of a job execution, workers notify the server, which collects global job statistics. Peers update their local donor list with information exchanged with other peers and by an explicit update message from the server (Fig. 5). In general, the local list of potential donors can be limited to contain only a subset of all peers (the top elements) and does not need to be strictly updated and synchronized in the system. Namely, for a very large system, the centralized server might become a bottleneck and, thus, might send less frequent updates to

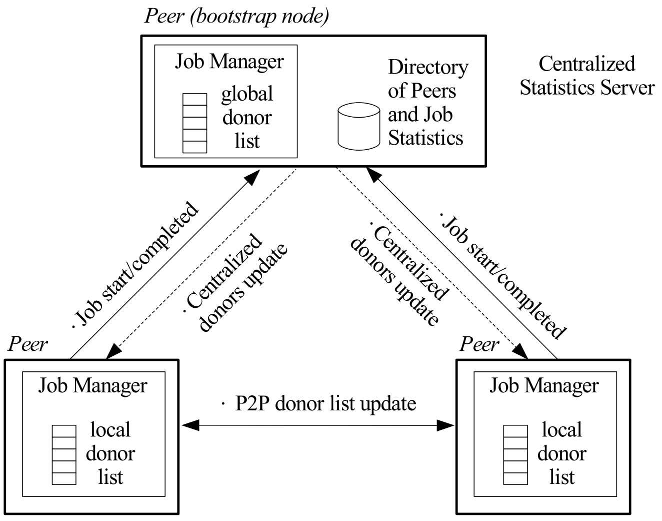


Fig. 5. Hybrid centralized/decentralized communication system for donor selection.

the peers (dashed line in the figure). However, the centralized server is not so critical in our approach since job statistics are also exchanged between peers.

The centralized server at the bootstrap node allows the determination of suitable job donors from the complete knowledge of the job statistics in the system. Approaches based on global statistics are known to provide optimal load balancing performance, while randomized techniques provide high scalability. We expect that for a larger and larger number of peers the performance of the DLB would decrease in a graceful way to the lower bound given by a simple random polling policy.

In some aspects, our approach is similar to the one described in [5] as a "modified" scheduler-based load balancing. In the latter, donors directly assign subtasks to idle workers, as in our case. However, in the approach described in [5], the poll is always generated by the centralized server, which adopts a round robin donor selection among the workers. In our case, the generation of job requests (polls) has also been decentralized. Moreover, in our DLB approach the use of the ranked-random technique is fundamental for the selection of the most suitable donors to avoid the generation of a high number of trivial tasks. Then, in our case, polls are not uniformly distributed among all workers, but they are still spread over several donors.

3.3. Search termination detection

Each JM keeps a list of assigned and not completed jobs in order to support mechanisms for fault tolerance and termination detection. With the adoption of a global collection of job statistics is quite straightforward to detect the termination of the entire search task. When all peers are not executing a job, have an empty job pool and an empty list of assigned jobs, then the search tree has been explored entirely.

When the search termination condition is detected, the centralized server will issue a broadcast message to inform the peers in order to start the next phase, i.e. the reduction of all local lists of partial results.

In some cases it may not be convenient to adopt a centralized server for the termination detection. For a very high number of peers it may become a bottleneck and decrease the performance by simply delaying the start of the next phase. Also hierarchical systems, like multi-domain computational environments, may suffer from a centralized approach. In these cases a distributed algorithm [18] for the termination detection can be adopted.

3.4. Reduction of final results

Each *Worker* maintains only a local and partial list of results found during the execution of subtasks. Therefore, at the end of the search process we perform a reduction operation. Workers are organized in a communication tree and the number of communication steps required is in the order of O(log$N$), where N is the number of processes. Generic collective operations, including the reduction, have been implemented in the CM and can be specialized for the specific application by the *Worker*.

This is more efficient than a star topology of a master-slave approach (e.g. [19]), which requires O(N) sequential communication steps.

In the test application, partial local lists of frequent fragments are merged and duplicate and non-discriminative fragments are discarded. However, the determination and selection of the discriminative fragments include expensive graph and subgraph isomorphism tests and may represent a non-trivial computational cost for a single processor. Moreover, the number of all frequent fragments may become very large and the communication cost would be too expensive. Therefore, the selection of the discriminative fragments has to be distributed as well. This can be performed during the reduction operation in parallel by several concurrent processes and not only by the single node where the results will be finally collected.

The reduction operation based on a logical communication tree has the advantage of a minimum number of communication steps. Moreover, it also has the advantage of distributing the computational load associated to the costly graph and subgraph isomorphism tests for the reduction of the final list of the molecular fragments.

4. Performance evaluation

The distributed application has been tested for the analysis of a set of real molecular compounds – a well-known, publicly available dataset from the National Cancer Institute, the DTP AIDS Antiviral Screen dataset [20]. This screen utilized a soluble formazan assay to measure protection of human CEM cells from HIV-1 infection [21]. We used a total of 37,169 compounds, of which 325 belong to the class of active compounds and the remaining 36,844 to the class of inactive compounds.

Experimental tests have been carried out on a network of Linux workstations[1] located in different LANs of the University of Konstanz; the software has been developed in Java. The communication among processes has been implemented using TCP socket API and XML data format.

In our tests, we introduced a synchronization barrier to wait for a number of processors to join the P2P system before starting the mining task only in order to collect performance results. In general, this is not necessary, but in the following results we did not want to consider the latency that is required to simply start up the remote peers.

4.1. Dynamic load balancing performance

In order to evaluate our DLB algorithm (RRP), we also implemented a random polling (RP) policy with distributed job queues and a centralized job-pool approach, i.e. a master-slave (MS) architecture [19]. The three techniques use the same work-splitting mechanism described in [17].

We compare three performance figures, the running time, the Jain's fairness index [22] and the relative load imbalance index. The running time gives an immediate measure of the effectiveness of the DLB. The Jain's fairness index is often adopted to measure the equality of the allocation of a shared resource to different contending entities. In this case, we want to measure the quality of the load balancing, which aims at the equal allocation of the overall computational load to processors. The Jain's fairness index is defined as:

$$J = f(x_1, x_2, \ldots, x_N) = \frac{\left(\sum_{i=1}^{N} x_i\right)^2}{N \cdot \sum_{i=1}^{N} x_i^2}, \tag{1}$$

where x_i is a measure of the work load at processor P_i. In particular, the quantity we consider is the running time spent in useful work by each processor:

$$x_i = T_{P_i,\text{work}}.$$

The index of formula (1) is continuous and bounded in the range $[0, 1]$. It is independent of the scale, the metric and the total amount of the shared resource and of the number of contending entities. An index closer to 1 means a better fairness in the allocation, hence better load balancing.

Another index that has often been adopted to measure the DLB performance (e.g. in [23]), is the relative load imbalance index (LI), given by

$$\text{LI} = f(x_1, x_2, \ldots, x_N) = 1 - \frac{\sum_{i=1}^{N} x_i}{N \cdot \max_{i=1}^{N} x_i}. \tag{2}$$

[1] Nodes have different hardware and software configurations. The group of the eight highest performing machines is equipped with a CPU Intel Xeon 2.40 GHz, 3GB RAM and run Linux 2.6.5-7.151 as well as Java SE 1.4.2_06.

Table 1
Dynamic load balancing performance (*minSupp* = 6%, *maxSupp* = 1% *and* $N_{\text{procs}} = 24$)

	MS	RP	RRP
Runtime (s)	91	114	76
Jain's index	0.90	0.99	0.99
LI index	0.38	0.10	0.13

The index is bounded in the range [0, 1], but it does not have the other properties described above for the Jain's index. The LI index is a measure of the load imbalance; a lower value means better load balancing.

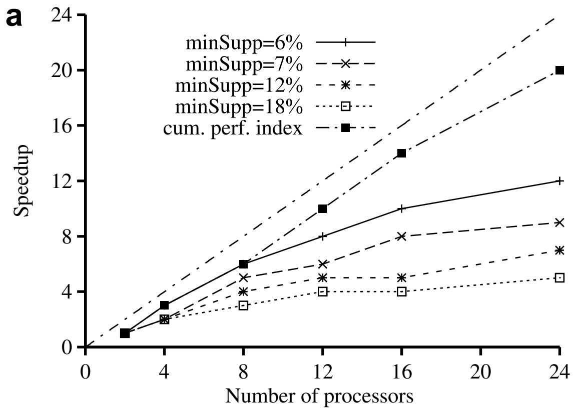
Table 1 provides a comparison of typical values of these three performance figures for the different DLB algorithms. In general, the two random approaches attain better load balancing because they distribute job requests to more donors than the centralized approach, which adopts a single job-pool at the manager and a single donor. However, the simple random polling is not able to reduce the running time because of the irregularity of the search space. A more accurate selection of donors (cf. Section 3.2) still allows for a better distribution of the work load, while it also attains a lower running time.

It should be noticed that both indexes are not sufficient to evaluate the quality of DLB policies; the running time must be taken into account. According to the LI index RP should perform slightly better than RRP, while it actually shows longer running time. The reason is that the LI index puts in evidence small differences that are not relevant. On the contrary, the Jain's fairness index provides indications that are, at least, not in contradiction with the running times. It is saturated (almost 1.0) in both the randomized techniques and their slight difference is not in evidence.

4.2. Speedup

We complete the analysis of the distributed application with the ranked-random DLB technique by showing the running time and the speedup curve (Fig. 6) of the parallel over the serial algorithm. The performance of the sequential algorithm that is used to determine the speedup, is obtained from an optimized version of the algorithm described in [14] in the highest performing single-processor[2] server that is available for our tests. Moreover, we carried out a series of preliminary tests to measure the performance of each available workstation for the specific mining algorithm. A cumulative performance index is shown in the figure and provides a speedup bound for the different configurations.

The speedup curves show that for lower *minSupp* values, i.e. more complex mining tasks, the speedup improves, as expected. In particular, for the value *minSupp* = 6%, the speedup is linear and exactly follows the speedup bound



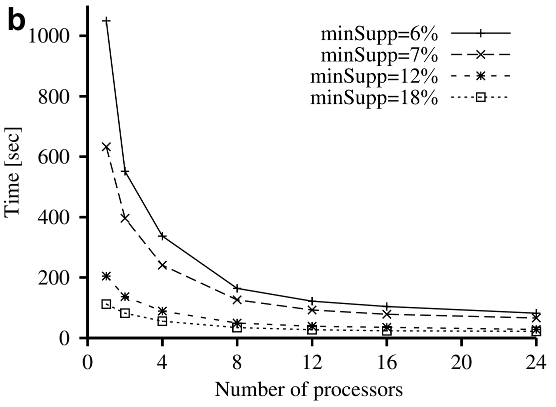


Fig. 6. (a) Speedup and (b) running time (maxSupp = 1%).

in the first part of the chart. Then, it is evident that more resources cannot further decrease the running time; the amount of work is not enough and additional computational resources cannot be effectively exploited. Nevertheless, it is positive that the running time does not increase when unnecessary resources are used as one might expect because of the additional communication and computation overheads. This provides evidence of the good scalability properties of the system. Moreover, it shows that the determination of the optimal number of processors for a given task is not critical with respect to the running time.

5. Conclusions

In this paper we have presented a distributed computing framework for problems based on a search strategy. It employs a decentralized dynamic load balancing technique that is enhanced by global statistics to cope with highly irregular problems. The proposed approach is particularly useful for parallel and distributed applications where no assumption of uniform or bounded task times can be made and no workload estimate is available. For these application simple DLB policies, like random polling and scheduler-based, do not provide an efficient solution.

We adopted the distributed computing framework for a parallel formulation of the frequent subgraph mining prob-

[2] Intel P4 3 GHz 2 GB RAM.

lem applied to the task of discriminative molecular fragment discovery.

Experimental tests on real molecular compounds in a distributed non-dedicated computing environment confirmed its effectiveness in terms of running time performance, low parallel overhead and load balancing. In a 24-node system and in comparison to a Master/Slave approach, the improvement of the running time was up to 16% and the quality of the load distribution (Jain index) improved up to 10%. In larger computational environments we expect that such improvements become more important.

Future research efforts will focus on a more accurate formalization of the model and its simulation on very large-scale systems. Experimental tests on large-scale systems are needed to verify the scalability of the P2P approach. The centralized server for job statistics could potentially become a bottleneck and a completely decentralized solution should be adopted.

Other research directions include the adoption of the approach in other application domains to verify and extend its general applicability and the introduction of advanced and intelligent services, e.g. the dynamic and autonomous management of overlay networks of peers. Peers can dynamically organize themselves in clusters in order to aggregate computing resources for computing-intensive subtasks by exploiting physical connectivity. This would be particularly useful for those problems whose complexity is not known in advance or it is difficult to be estimated with sufficient precision. At run time further resources could be dynamically aggregated and allocated to particularly difficult subtasks.

Acknowledgements

This work was partly supported by the Deutsche Forschungsgemeinschaft (DFG) Research Training Group GK-1042 "Explorative Analysis and Visualization of large Information Spaces". We thank the Department of Computer and Information Science of the University of Konstanz for the use of their machines.

References

[1] R. Finkel, U. Manber, Dib – a distributed implementation of backtracking, ACM Transactions on Programming Languages and Systems 9 (2) (1987) 235–256.

[2] R. Karp, Y. Zhang, A randomized parallel branch-and-bound procedure, in: Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC), 1988, pp. 290–300.

[3] S. Chakrabarti, A. Ranade, K. Yelick, Randomized load-balancing for tree-structured computation, in: Proceedings of the Scalable High Performance Computing Conference (SHPCC), 23–25 May 1994, Knoxville, TN, 1994, pp. 666–673.

[4] Y. Chung, J. Park, S. Yoon, An asynchronous algorithm for balancing unpredictable workload on distributed-memory machines, ETRI Journal 20 (4) (1998) 346–360.

[5] V. Kumar, A. Grama, V.N. Rao, Scalable load balancing techniques for parallel computer, Journal of Parallel and Distributed Computing 22 (1) (1994) 60–79.

[6] T. Washio, H. Motoda, State of the art of graph-based data mining, ACM SIGKDD Explorations Newsletter 5 (1) (2003) 59–68.

[7] R. Agrawal, T. Imielinski, A.N. Swami, Mining association rules between sets of items in large databases, in: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, 26–28 May 1993, Washington, DC, 1993, pp. 207–216.

[8] M.R. Garey, D.S. Johnson, Computers and intractability: a guide to the theory of NP-completeness, W.H. Freeman, 1979.

[9] S. Skiena, Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, Addison-Wesley, 1990.

[10] E.M. Luks, Isomorphism of graphs of bounded valence can be tested in polynomial time, Journal of Computer and System Sciences 25 (1982) 42–65.

[11] M. Deshpande, M. Kuramochi, G. Karypis, Frequent sub-structure-based approaches for classifying chemical compounds, in: Proceedings of IEEE International Conference on Data Mining (ICDM), 19–22 November 2003, Melbourne, Florida, USA, 2003.

[12] G. Di Fatta, A. Fiannaca, R. Rizzo, A. Urso, M.R. Berthold, S. Gaglio, Context-aware visual exploration of molecular databases, in: Proceedings of the International Workshop on Data Mining in Bioinformatics, 6th IEEE International Conference on Data Mining, 18–22 December 2006, Hong Kong, 2006.

[13] M. Zaki, S. Parthasarathy, M. Ogihara, W. Li, New algorithms for fast discovery of association rules, in: Proceedings of 3rd International Conference on Knowledge Discovery and Data Mining (KDD), 1997, pp. 283–296.

[14] C. Borgelt, M.R. Berthold, Mining molecular fragments: Finding relevant substructures of molecules, in: IEEE International Conference on Data Mining (ICDM), 9–12 December 2002, Maebashi, Japan, 2002, pp. 51–58.

[15] X. Yan, J. Han, gSpan: graph-based substructure pattern mining, IEEE International Conference on Data Mining (ICDM) (2002) 721.

[16] M. Deshpande, M. Kuramochi, G. Karypis, Automated approaches for classifying structures, in: Proceedings of Workshop on Data Mining in Bioinformatics (BioKDD), 2002, pp. 11–18.

[17] G. Di Fatta, M.R. Berthold, High performance subgraph mining in molecular compounds, in: Springer's LNCS Proceedings of the International Conference on High Performance Computing and Communications (HPCC), 21–24 September 2005, Sorrento, Italy, 2005.

[18] C. Sieb, G. Di Fatta, M.R. Berthold, A hierarchical distributed approach for mining molecular fragments, in: Proceedings of the ECML/PKDD 2006 Workshop on Parallel Data Mining, 18–22 September 2006, Berlin, Germany, 2006.

[19] G. Di Fatta, M.R. Berthold, Distributed mining of molecular fragments, Proceedings of the Workshop on Data Mining and the Grid (DM-Grid) of the IEEE International Conference on Data Mining (ICDM), 1–4 November 2004, Brighton, UK, 2004.

[20] National Cancer Institute, DTP AIDS antiviral screen dataset. <http://dtp.nci.nih.gov./docs/aids/aids_screen.html/>.

[21] O. Weislow, R. Kiser, D. Fine, J. Bader, R. Shoemaker, M. Boyd, New soluble formazan assay for hiv-1 cytopathic effects: Application to high flux screening of synthetic and natural products for aids antiviral activityJournal of the National Cancer Institute, 81, University Press, Oxford, United Kingdom, 1989, pp. 577–586.

[22] R. Jain, D. Chiu, W. Hawe, A quantitative measure of fairness and discrimination for resource allocation in shared computer systems, Technical report, DEC Research Report TR-301 (1984).

[23] R. Sakellariou, J.R. Gurd, Compile-time minimisation of load imbalance in loop nests, in: Proceedings of the 11th ACM International Conference on Supercomputing (ICS), July 1997, pp. 277–284.