Distributed Branch-and-Bound Algorithm : A Pure Peer-to-Peer Approach
Mathieu Djamai∗ Bilel Derbel∗ Nouredine Melab∗

Résumé
The state-of-the-art large scale approach for solving NP-hard permutation-like problems using parallel Branch-and-Bound (B&B) techniques are based on a Farmer-Worker model which is known to be limited in terms of scalability. In this paper, we propose a new pure Peer-to-Peer (p2p) approach for parallel B&B. Our approach can efficiently manage an exponential amount of computational entities in a fully distributed way. In fact, we give simple and efficient algorithms working in a fully distributed manner and dealing with major parallel B&B issues such as global information sharing, dynamic load balancing and termination detection. All our algorithms are formally proved to be correct. From the experimentation side, our approach was implemented and validated on top of the Grid’5000 nation-wide French grid. For the well-known Flow-Shop scheduling problem, it permits to (i) improve the parallel efficiency of the state-of-the-art Farmer-Worker approach, (ii) to gain up to 10% in terms of number of solutions explored during the B&B search process, and last but not least,(iii) to keep the communication overhead low.

Keywords : Peer-to-Peer, Branch and Bound, Grid, Distributed Algorithms, Overlay Networks, Termination Detection.

1 Introduction

Nowadays, more and more computational resources are available to tackle highly resource consuming problems as in the field of Combinatorial Optimization. This class of problems admit, for larger instances, a huge number of possible solutions and require the design of efficient software platforms to be solved optimally. Designing such a platform is not straightforward as it has to take advantage from both large-scale computational grids and personal computers connected through Internet. Most of the existing platforms ([BMT09] [AFH02] [Cun94]) propose a Master-Slave architecture: A set of entities called "slaves" are in charge of computing tasks related to the application and obtain the results and the coordination of this set of computational entities is done through the "master" entity. However, this architecture is known to have a limited scalability because of the synchronization operations and the communications bottleneck around the master.

In this paper, we choose to design a fully distributed/pure P2P architecture for the Branch-and-Bound algorithm to solve these issues and gain scalability. Several challenges have to be faced while evolving in a distributed environment. As no central entity is available any more, our algorithms must operate only with local information. Network communication have to be kept low. Synchronization operations and time delays must have a negligible impact on the overall performances. Thus, we redesign B&B mechanisms such as work sharing, dynamic load balancing, termination detection combined with a pure P2P architecture, namely a Pastry-like [RD01] [MKL+08]. Our approach avoids redundancy during exploration process and can manage a number of entities exponentially higher than a traditional Master-Slave one. We validated experimentally our approach on the French nationwide...
Grid’5000 experimental grid of the Flow-Shop Scheduling Problem, resulting, compared to the Mez-maz et al.’s Master-Slave approach [MMT05], in a better and non-decreasing Parallel Efficiency rate of 98% contrarily to the latter, a gain up to 10% in terms of solutions explored and a low communication overhead in terms of messages exchanged.

The rest of the paper is organized as follows. In Section 2, we give some backgrounds concerning the parallelization of the B&B algorithm. In Section 3, we describe our Peer-to-Peer approach and analyze it. In Section 4, we provide experimental results on an instance of the Flow-Shop problem (T057), comparing our approach with the state-of-art one. Section 5 draws some conclusions and open questions raised by our approach.

2 Background

From a parallel/distributed point of view, the most efficient B&B algorithms are based on a depth-first strategy when exploring the search space. For instance, for a permutation-like problem, the search space can be represented by a tree where a leaf represents a solution (a permutation) and a node inside the tree represents a partial solution (equivalently, a sub-problem) where only some variables in the permutation are fixed. Roughly speaking, a sequential depth-first strategy will start exploring the tree in a depth-first manner branching and bounding when necessary. A parallel version of this strategy is to run several depth-first exploration processes in parallel on different parts of the tree. As soon as a new lower bound for the problem is found, it is communicated to other processes which permits them to update their own local lower bound and thus to speed up the overall search process. One major difficulty when setting up this general idea in a distributed environment is to carefully encode the node of the B&B tree in order to reduce the cost of messages exchanged between computational nodes and decide which part of the tree should be explored by which node. A trivial approach would be to encode a sub-problem of the B&B tree in a fully-comprehensive way by providing all the necessary information. In [IF00], a tree node (a subproblem) is encoded as a path from the root to the node itself which is still expensive. In [MMT05, MMT07], an extremely simple and efficient encoding is proposed. We propose hereafter a brief explanation of the Master-Slave paradigm as well as a short description of this encoding as we will use it in our approach.

Fig. 1 shows a simple example with a 3-variable permutation. A number is assigned to each leaf of the tree, corresponding to a specific permutation. Intervals, which can be considered as work units, represent subsets of the entire search space (the leaves of the tree). Concretely, an interval is associated to a set of leaves and all the nodes contained in the paths from the leaves towards the root of the tree. Two operators permit to switch from the tree representation to the interval-based representation, called

2. the root of the tree represents the whole problem, that is no variable is fixed yet at the root.
Distributed Branch-and-Bound Algorithm:
A Pure Peer-to-Peer Approach

respectively "Fold" and "Unfold".

In a Master-Slave architecture, the slaves ask for an interval to the "master". The latter, managing the global interval (corresponding to the whole problem), splits it and attributes one of the two sub-intervals to the requester. This system may engender redundancy in the exploration of solutions since if a node is temporarily unavailable, its interval can be, in the meantime, reassigned to another slave. It ends up with two slaves exploring the same solutions.

3 Our Pure Peer-to-Peer B&B approach

In the Master-Slave approach, the architecture can be viewed as a star graph. The center of the star (the master) has a global view of the remaining intervals to be explored, and can communicate directly with the leaves (the slaves), easing the coordination of the parallel B&B tree exploration. As pointed earlier, it induces synchronization issues and a communication bottleneck around the master. So to gain in scalability, the idea of our approach is to evolve towards a pure P2P architecture by redesigning the mechanisms linked to the B&B algorithm. In fact, handling more computational nodes organized according to a pure P2P overlay implies more communications and message exchanges to coordinate the B&B search process which could lead to an efficiency loss if not managed very carefully. In the following, we describe how to proceed in a static environment, i.e. when computational nodes are fully available and do not crash.

3.1 Information sharing and work distribution

Following the interval based approach of [MMT05], there are two main issues to handle when exploring the B&B tree in parallel. Firstly, each computational node should be assigned an interval to explore. To guarantee that the optimal solution will effectively be computed, we must ensure that an interval is assigned to at least one node. In addition, two intervals assigned to two nodes should not intersect, to avoid exploring the same region of the search space and thus losing efficiency. Secondly, each computational node should be aware of the best solution found so far during the search process in order to avoid exploring unwanted branches, i.e., branches not leading to the optimal solution. In fact, the faster a node is aware of the best solution the more the B&B branching is efficient. Having these issues in mind, we now describe our distributed algorithms.

The best solution sharing is done through a simple rule. If a node finds an improving solution or is informed by a neighbour about a better solution than the one it knows, it spreads it to its neighbours. Thus, these neighbours can eventually update their local best-known solution and branch partially their exploration sub-tree. Concerning the main guidelines of how the work is shared between nodes, we assume that initially there exists exactly one node in the network holding the interval $[0,N]$ where $N$ is the size of the problem being solved, i.e. for a permutation of size $p$, we have $N = p!$.

Then, the nodes having initially no interval to explore or those that have finished exploring some intervals, asks their neighbours for a piece of interval to explore. Upon receiving such a request, a node $v$ holding an interval $[x,y]$ being explored halts the searching process. If the node $v$ is exploring the branch corresponding to an integer $z \in [x,y]$, then it replies with interval $[(z+y)/2,y]$ and continues its local exploration process using interval $[z,(z+y)/2]$.

Clearly, this work sharing mechanism guarantees that no kind of redundancy will occur. More precisely, an interval will not be explored by two processors at the same time, since a processor, to satisfy a work request, only sends an interval it hasn’t explored yet. However, it may happen that at the time a node $v$ sends a request, none of its neighbors has an interval to share with him. As we will see
in the next section, this issue is to be handled tightly to the termination detection of the B&B search process.

3.2 Distributed Termination Detection

When a node \( v \) asks its neighbours for an interval, it may be either successful or vain. So, the termination is to be detected only using this local information as no global information is available. To achieve this goal, we use the key observation that an unsuccessful request implies that the neighbours are symmetrically waiting for some work. However, basing our mechanism only on the work requests shall lead to a deadlock situation in the termination phase where all the nodes begin asking each other for an interval and nobody answers since the B&B process is terminated.

To correctly detect the global termination of the B&B search process, we turn out to a technically involved solution. Generally speaking, we remark that by repeating the local request/response process \( t \) times, a node can detect that no intervals are available to other nodes in the ball of radius \( t \) around it. Therefore, after \( t = D \) unsuccessful rounds, where \( D \) is the diameter of the overlay graph, a node can locally conclude that no interval is being explored by any node in the network and termination can thus be detected. Technically speaking, we use an adaptation of this idea taking into account the specificity of the B&B exploration process in order to correctly detect termination. More specifically, every node \( v \) owns a variable \( \text{count}_{req} \) counting the number of times the node asks its immediate neighbors for an interval. When the node finishes exploring its interval, its counter is at 0 and asks its neighbours for an interval and their counter. Every time such a request is unfruitful, the counter is incremented else a new interval to explore is given to the node and resets its counter. If the counter reaches \( D \) then the termination is detected.

According to this scheme, if a node \( v \) in the network holds an interval, this interval will be shared and spread itself to the other nodes after a definite number of iterations. This interval acts like a message informing the nodes that there is still some work to accomplish. However, a particular case has to be taken into account.

A case that can occur is, while only one node has an interval to share, the interval reaches a node \( w \) and this node ends the exploration of this interval before it shares it with a neighbour \( y \) who is still waiting for some work. This phenomenon can be called "interval vaporization". This neighbour \( y \) and all other nodes waiting for work have to be informed that there is still some work to accomplish. It is necessary to keep spreading the information carried by the intervals. This information is still present in the fact that a difference appears between counters of nodes \( w \) and \( y \) and this difference means that an interval is present at a distance comprised between \( \text{count}_{req}_w \) and \( \text{count}_{req}_y \). This is why node \( y \) must reset its counter \( \text{count}_{req}_y \) to the lower value \( \text{count}_{req}_w \) to keep propagating the information about the presence of an interval.

Remark : As it iteratively sends request to all neighbours, it is clear that to detect the termination or the presence of an interval somewhere in the network, in a bad scenario, it will send about \( O(d_v \cdot D) \) messages with \( d_v \) the degree of node \( v \).

Proposition 1. Consider that, at a given time, a computational node \( v \) has no work interval to explore. In the worst case, it sends at most \( O(d_v \cdot D) \) messages on the network until either it receives a work unit or it detects the termination of the B&B algorithm.

3. Notice however that, at this stage, every node will own the optimal solution, i.e., the B&B algorithm is globally terminated but no node can locally detect that the search is actually terminated.
4. That is after \( D \) local requests without getting an interval to explore.
5. \( d_v \) is the degree of node \( v \).
Distributed Branch-and-Bound Algorithm: 
A Pure Peer-to-Peer Approach
Djamai, Derbel and Melab

As we tackle large Flow-Shop problem instances, new improving solutions are found quite scarcely through execution time so we can reasonably state that the communication complexity of our approach is dominated by the work sharing mechanisms. Thus, to gain in efficiency, the overlay graph should have a good balance between node degrees (i.e. each peer must have a low number of neighbours) and network diameter (i.e., peers must stay relatively close in the logical overlay). In fact, spreading a piece of information (intervals) over the network must be performed quickly while keeping the communication load of the peers at a low level to avoid bottlenecks around peers. To satisfy these constraints, we make use of the well known Pastry-like overlay [RD01] where both the diameter of the graph and the average degree are in \( \log_2 k (n) \) with \( n \) the number of nodes.

Assuming the latter overlay topology, and using Proposition 1, the combined work sharing and termination detection distributed algorithm described before now costs at most \( (\log_2 (n))^2 \) messages per node. To illustrate how this measure improves the Master-Slave approach of [MMT05] in terms of scalability, let us assume that the limitation of the Master-Slave approach is due to the bottleneck phenomenon. In the Master-Slave approach, in order to share work or to announce termination, the number of messages sent on the network is thus \( O(n) \) with the "master" having a degree equal to \( n \). In the Peer-to-Peer approach, for a number \( n' \) of peers, the number of messages sent arises to \( O(n' \cdot (\log_2 (n'))^2) \). Let \( n \) be the maximum of machines that can be handled by the Master-Slave approach, that is \( n \) is now the maximum number of workers that leads to a bottleneck around the master. In the Peer-to-Peer approach, \( n \) would be also the maximum number of tolerated neighbors before a communication bottleneck around each peer can be observed. Thus, if every peer has a degree equal to \( n \) and \( n' \) is the resulting number of peers in the network, we have that up to some constant \( n = \log_2 (n') \) which means that the maximum number of nodes allowed by our scheme is \( n' = 2^{kn} \). This shows that despite an overhead of \( (\log_2 (n'))^2 \) in terms of messages sent, which is low, our approach allows to improve exponentially the scalability of the Master-Slave approach. As our experimental results show in section 4, we even obtain a lower communication overhead than expected theoretically while being more efficient in terms of space exploration and parallel efficiency.

4 Experimental results

This section presents some experiments conducted on the French Nationwide Grid’5000 experimental Grid. Our algorithms described in previous sections were validated on all the instances of the Flow-Shop Scheduling Problem from which the optimal solution is known, at the time of writing. Several implementations of Pastry are available (such as FreePastry [ ?]). We choose to develop our own implementation and adapt it to our problem. As work units are coded using the method designed by Mezmaz et al., the Peer-to-Peer approach’s performances will be compared to the Mezmaz’s Master-Slave one. Each experiment lasted one hour. Every obtained value is an average on ten independent runs. Three different sites have been used, located in Bordeaux, Orsay and Nancy and no specific environment has been deployed on the computational nodes. Efforts have been made to distribute equally the peers among the three sites used for the experimental runs. The main challenge is to obtain a good parallel efficiency, namely for a given execution time, to ensure that the machines spend more time to compute in the P2P approach than in a Master-Slave one, by avoiding as much synchronization operations and communications with a central entity.

---

6. The \( 2^k \) logarithm’s basis is due to the neighborhood definition and \( k \) stands for the maximum number of characters a peer ID may contain.

7. Here, we implicitly assume that scalability limitations in [MMT05] is not due to network congestion which is actually the case as illustrated later by our experiments.
As shown in Fig. 2, which compares both the Parallel Efficiency rate (graph A) and the Communication rate (graph B) to the number of peers, two points can be emphasized. On one hand, the Parallel Efficiency (PE) of the P2P approach maintains itself at a higher rate than Master-Slave’s rate. On the other hand, this PE rate does not decrease for the P2P approach, contrarily to the Master-Slave’s. On graph A, the standard deviations are 0.23\% for the "P2P" set of data and 1.41\% for the "Master-Slave" one. For graph B, these values are respectively 0.11\% and 1.24\%.

The explanation of this phenomenon is rather simple. In a Master-Slave approach, when the number of workers increases, so does the communication load on the Master entity. Thus, communication delays become more and more important, this is why the Master-Slave PE rate decreases on Fig. 2a. In the P2P approach, the communication load is distributed among all the peers so the load on each peer remains almost constant since as shown theoretically in section 3, an exponential number of peers than can be handled by the network. It is interesting to notice that this reduced communication load observed confirms the idea exposed earlier which is that the scalability is increased despite an overhead in terms of messages exchanged over the network as shown in Fig. 3a. (In our case, for each experiment, the number of peers involved as well as the network topology are fixed. Usually, the parameter \( k \) is generally set to values higher than 4 for Internet-wide applications. Here, due to the grid context and the predefined Pastry-like organization of the peers in the network, the most accurate value for parameter \( k \) is 2.)

The "theoretical" curve on Fig. 3b represents the maximal message overhead that can be expected theoretically (i.e. namely the "worst case" scenario). The "experimental" curve shows that the peers are actually far from being overloaded in terms of communication.

For a very low number of cores (beneath 70), the network communications are not a real constraint to manage for the network. This is why performances between both approaches are identical and maximal. Benefits of our approach do not show up at such small scales as, in Fig. 3a, the speedup appears only when using at least approximatively 70 peers.

These performances can also be seen through the gain in terms of solutions explored by both approaches. The number of solutions explored includes both the number of solutions evaluated (the leaves of the search tree) and the number of solutions eliminated by the branching operator of the algorithm. The gain showed in Fig. 3a is obtained by comparing the number of explored solutions by the P2P approach to the same number in the Mezmaz’s Master-Slave approach.

---

8. The "Theoretical" curve represents the function \( n \mapsto (\log_{2}(n))^2 \) as theoretically, the overhead was \((\log_{2}(n))^2\)
5 Conclusions and Future Works

In this paper, we have proposed a new fully Peer-to-Peer approach for the B&B algorithm parallelization with new algorithms allowing to efficiently tackle the problems linked to the communication bottleneck and the synchronization operations. (The solution coding method is inspired from Mezmaz et al.’s Master-Slave approach.) The chosen topology allows to obtain an average node degree and a network diameter of both $O(\log_2(n))$, with $n$ standing for the number of nodes, and $k$ the maximum number of characters a peer ID can contain. Thus, the approach is able to handle a number of nodes which is exponentially higher than a classical Master-Slave one, as shown both theoretically and experimentally. Moreover, it provides a new work sharing mechanism where each peer shares its interval with its neighbours on request, avoiding redundancy during the tree exploration. It provides a distributed termination detection mechanism which detects the presence (or absence) of a work unit somewhere in the network by counting the number of unsuccessful requests broadcasted to its neighbours. It is planned to extend this approach to dynamic environments (i.e. where resources are volatile) and apply it to other tree-based discrete optimization exact methods. The objective is to design a generic Peer-to-Peer platform for distributed systems.

Références


Distributed Branch-and-Bound Algorithm:
A Pure Peer-to-Peer Approach

Djamai, Derbel and Melab
