Journal of King Saud University – Computer and Information Sciences (2015) **27**, 376–385

King Saud University

Journal of King Saud University – Computer and Information Sciences

www.ksu.edu.sa
www.sciencedirect.com

Server consolidation for heterogeneous computer clusters using Colored Petri Nets and CPN Tools

Issam Al-Azzoni

College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia

Received 14 December 2014; revised 26 January 2015; accepted 15 February 2015
Available online 9 October 2015

KEYWORDS

Colored Petri Nets;
Heterogeneous clusters;
Server consolidation;
Discrete optimization;
CPN Tools;
State space search

Abstract In this paper, we present a new approach to server consolidation in heterogeneous computer clusters using Colored Petri Nets (CPNs). Server consolidation aims to reduce energy costs and improve resource utilization by reducing the number of servers necessary to run the existing virtual machines in the cluster. It exploits the emerging technology of live migration which allows migrating virtual machines between servers without stopping their provided services. Server consolidation approaches attempt to find migration plans that aim to minimize the necessary size of the cluster. Our approach finds plans which not only minimize the overall number of used servers, but also minimize the total data migration overhead. The latter objective is not taken into consideration by other approaches and heuristics. We explore the use of CPN Tools in analyzing the state spaces of the CPNs. Since the state space of the CPN model can grow exponentially with the size of the cluster, we examine different techniques to generate and analyze the state space in order to find good plans to server consolidation within acceptable time and computing power.

1. Introduction

Advances in virtualization has enabled server consolidation in grid and cluster computing systems. Using virtual machines to run the applications of an enterprise, it is possible to reduce the number of machines in a cluster by grouping together multiple virtual machines in one physical machine. This allows better multiplexing of cluster resources across applications.

To deal with the dynamic workloads that characterize cluster applications, it is necessary to use dynamic server consolidation techniques. Dynamic server consolidation techniques employ live migration of virtual machines (VMs). Using live migration, VMs can be reallocated across the physical machines (PMs) to improve the manageability of clusters (Clark et al., 2005). However, it is important to reduce the migration overhead, i.e., the amount of data transferred, since this has an impact on the performance of the applications that are run by the migrating VM (Wood et al., 2007).

Several approaches to dynamic server consolidation for computer clusters have been proposed (see Bobroff et al. (2007) and Hermenier et al. (2009)). These approaches aim to minimize the number of PMs necessary to host the VMs. However, most of these approaches neglect the ensuing migration overhead. While server consolidation is conceptually

E-mail address: ialazzoni@ksu.edu.sa
Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.jksuci.2015.02.001

close to the classical vector-packing problem which is NP-hard, there are several factors which introduce extra difficulties. These include the need to minimize the migration overhead and the need to deal with the possibly heterogeneous physical machines. Moreover, server consolidation must respect different constraints on the placement of the VMs.

In this paper, we propose a new approach to consolidation that is based on Colored Petri Nets (CPNs) (Jensen and Kristensen, 2009). CPNs provide a graphical language to describe and analyze models of concurrent systems. CPNs have been used in a wide variety of domains, however this paper is the first to recommend their use in server consolidation for heterogeneous clusters. As a modeling language, CPNs naturally model server consolidation problems and thus our aim is to explore their use as a solver method. In the CPN model, we use places to model the PMs and colored tokens to model the VMs. The CPN model is designed to capture knowledge on the resources of each individual PM. By doing so, our approach has the advantage of being applicable to heterogeneous clusters. At the same time, our approach incorporates the constraints on the VM placement directly in the CPN model.

We use CPN Tools to create the CPN models and analyze their state spaces. Since the state space of the CPN models developed in this paper can grow exponentially, we propose several techniques to reduce the size of the state space and to efficiently explore it.

The contributions of the paper are summarized as follows:

1. We explore the use of CPNs in the context of server consolidation.
2. The proposed approach not only minimizes the number of servers required to host the VMs, but also minimizes the required reconfiguration cost.
3. The proposed approach works for heterogeneous clusters and can incorporate several types of constraints on the placement of VMs.
4. We experiment with different techniques to deal with the resulting large state spaces allowing to scale the approach to work on larger cluster sizes.

The organization of the paper is as follows. In Section 2, we define the consolidation problem more formally. The related work is discussed in Section 3. We illustrate our approach in Section 4. In Section 5, we propose and analyze experimental results of several techniques to address the state explosion problem when applying our approach on larger clusters. Section 6 concludes the paper.

2. Problem formulation

Consider a cluster consisting of a number of physical machines. Each PM has a number of processing units and provides a certain amount of memory. Using virtualization technologies, a number of virtual machines are consolidated on the physical machines. Each VM requires a certain amount of memory. Depending on the state of the application(s) that are running on the VMs, a VM can be classified as an active or inactive VM. An active VM requires a number of processing units. On the other hand, an inactive VM does not require processing units. The cluster model is similar to Ferreto et al. (2011), Hermenier et al. (2009) and Murtazaev and Oh (2011).

A configuration of the cluster maps the VMs to the PMs. Starting from an initial viable configuration in which each VM has been allocated sufficient amount of memory and each active VM has access to its own required number of processing units, the objective is to reconfigure the cluster to reach a viable configuration that uses the minimum possible number of PMs. The reconfiguration steps must respect the VM requirements on the processing units and memory: a VM cannot be migrated if the destination PM does not have sufficient amount of free memory, and an active VM cannot be migrated if the destination PM does not have the required number of free processing units. The reconfiguration plan outlines the steps to reach the configuration which uses the minimum number of PMs. In order to reduce the migration overhead, the reconfiguration plan needs also to minimize the total amount of data transferred. The overhead for migrating a single VM is equal to its required size of memory (hereafter referred to as its memory size) .

Consider the initial configuration in Fig. 1. Each PM provides a single processing unit and has 1024 MB available memory. There are seven VMs with different memory sizes. Each active VM requires a single processing unit. Each VM is labeled with its memory size and state. The objective is to find a reconfiguration plan in order to reach a configuration that uses the minimum number of PMs. The reconfiguration plan should minimize the total data migration overhead.

Fig. 2 shows a possible minimal configuration which only requires three physical machines. The following outlines an optimal reconfiguration plan to reach the minimal configuration in Fig. 2:

1. Migrate VM5 from PM5 to PM4 (migration cost = 256 MB).
2. Migrate VM7 from PM1 to PM2 (migration cost = 384 MB).

The total migration cost for this plan is 640 MB. The resulting configuration only uses three PMs. Contrast this reconfiguration plan with another one computed using the First Fit Decreasing (FFD) heuristic, which has been used in other works on server consolidation (see Bobroff et al., 2007, Hermenier et al., 2009 and Wood et al., 2007). In FFD, VMs are first sorted by their memory sizes in decreasing order. Given the sorted sequence of VMs, each VM is migrated, if necessary, to the first (lowest indexed) PM into which it will fit (Coffman et al., 1997):

1. Migrate VM2 from PM3 to PM2 (migration cost = 512 MB).
2. Migrate VM1 from PM4 to PM1 (migration cost = 256 MB).
3. Migrate VM3 from PM2 to PM1 (migration cost = 256 MB).
4. Migrate VM5 from PM5 to PM3 (migration cost = 256 MB).
5. Migrate VM6 from PM4 to PM3 (migration cost = 256 MB).

The resulting configuration uses three machines similar to the optimal reconfiguration plan. However, the total migration cost for this plan is 1536 MB which is 240% more than that of the optimal reconfiguration plan. FFD does not consider

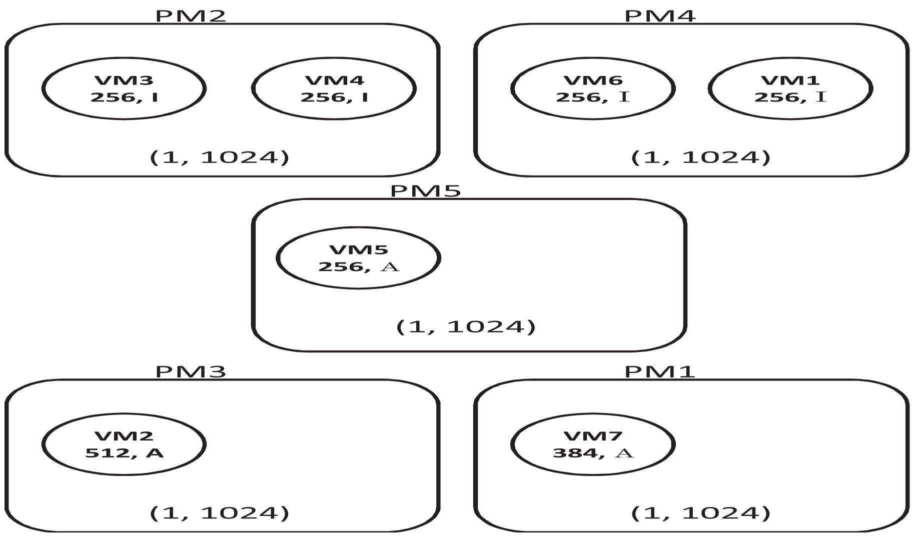


Fig. 1 An initial sample configuration for a five-machine homogeneous cluster. A semi-circular box represents a PM and a circle represents a VM.

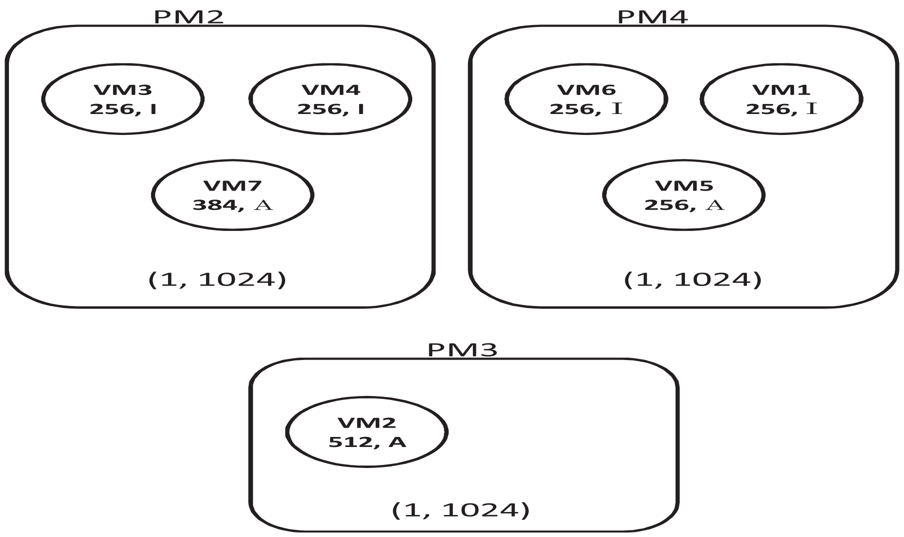


Fig. 2 A minimal configuration using three PMs only.

the initial placement of the VMs in the cluster. Also, FFD does not take the incurred migration overhead into consideration. These are the two main reasons explaining the significant total migration cost for the plan found using FFD.

Finding an optimal reconfiguration plan appears related to the NP-Hard 2-Dimensional Bin Packing Problem (see Caprara and Toth, 2001, Skiena, 2008 and Spieksma, 1994), where the dimensions correspond to the number of processing units and the amount of available memory. However, there are important differences between the two problems. In finding an optimal reconfiguration plan, the cluster starts with an initial viable configuration. The objective then is to determine the intermediate steps to reach a minimal viable configuration. The intermediate steps must respect the VM requirements on the processing units and memory (and possibly other constraints). These steps should also minimize the total VM migration overhead. These requirements increase the difficulty of the problem of finding an optimal reconfiguration plan.

Some virtual machines may not be consolidated on a given physical machine possibly due to missing platform requirements. In finding an optimal reconfiguration plan, one must restrict the intermediate configurations to satisfy these placement constraints. In Section 4, we discuss how the CPN model can be extended to incorporate such constraints.

3. Related work

Several approaches to server consolidation have been proposed. The authors of Hermenier et al. (2009) introduce Entropy which is a dynamic consolidation manager for homogeneous clusters. Consolidation in Entropy is based on constraint programing and takes migration overhead into account. Entropy uses two phases. The first phase finds the minimum number of nodes that are necessary to host all VMs and a sample viable configuration that uses this number of nodes. The second phase computes an equivalent viable configuration that minimizes the reconfiguration time. Entropy uses several techniques to reduce the computation cost, including the use of VM and PM equivalence classes. Entropy represents the closest consolidation manager to our approach. However, our approach directly considers heterogeneity of physical machines and can be extended to include the reconfiguration constraints discussed in Section 2.

The authors of Bobroff et al. (2007) propose a dynamic consolidation manager based on time series forecasting techniques and bin backing heuristics. The resource demands for the VMs, including CPU demands, can vary with time and thus the need to predict those demands for each forecast interval. Their reconfiguration algorithm which is based on FFD attempts to minimize the number of physical machines hosting virtual machines subject to the constraint that the rate of demand overloading the resource capacity is bounded by a specified threshold which is related to a Service Level Agreement (SLA). The VM migration overhead is not taken into account. Our approach assumes fixed resource demands for the VMs and considers the migration overhead.

ReCon (in Mehta and Neogi, 2008) is a planning tool that can be used to recommend server consolidations in multi-cluster data centers. Server consolidation aims to minimize the number of required PMs while satisfying system, application, and legal constraints. The problem is formulated in an optimization framework which is solved using CPLEX. Although the tool invokes the optimizer for each consolidation window to find an optimal step in the reconfiguration plan, the resulting reconfiguration plan incorporating all steps may not minimize the total migration overhead. Also, ReCon may not scale well for large data centers.

The authors of Ferreto et al. (2011) propose an LP formulation and heuristics to control VM migration. Their server consolidation approach does not migrate virtual machines with steady resource demands in order to reduce the effect of migration on the performance of their workloads. The approach is shown to reduce the number of virtual machine migrations and the required physical machines, however, the total data migration overhead is not taken into account. Thus, virtual machines with large memory demands are considered the same as those with small memory demands. Their approach is compared with a modified version of FFD which only migrates virtual machines with varying resource demands.

Sercon is another server consolidation algorithm (Murtazaev and Oh, 2011). The algorithm minimizes the number of required physical machines while attempting also to minimize the number of migrations. However, it does not consider the total data migration overhead neither the different memory demands of the migrated virtual machines (which our CPN approach does). Sercon algorithm inherits some properties of well-known algorithms for bin-packing, such as FFD, in addition to its aim to minimize the number of migrations. This is done by following an all-or-nothing property, that is all VMs from a node are migrated or if one of them fails, none of them are migrated. Using simulations, Sercon algorithm is compared with FFD in terms of migration efficiency which accounts for the number of VM migrations. The scalability analysis of Sercon demonstrates that the algorithm is scalable enough for middle-sized data centers (with VM numbers up to 1000 and PM numbers up to 100). The authors also propose a simple algorithm called Migration Ordering to improve the overall migration time. The idea is to group all the planned migrations into minimal number of queues of non-overlapping migrations. This algorithm can be applied in our context to reduce the overall migration time for an optimal reconfiguration plan since some steps in the plan can be done in parallel.

Our approach does not consider communication traffic patterns among VMs. In some situations, VMs with large mutual bandwidth usage are better consolidated in close proximity. The work of Meng et al. (2010) proposes a VM placement strategy that is traffic-aware. The authors formulate the VM placement as an optimization problem and design a heuristic that approximately solves the VM placement problem for large data centers. In addition, the work of Shrivastava et al. (2011) incorporates communication traffic patterns between VMs and knowledge on the underlying network topology into VM placement decisions for multi-tier applications.

In Jayasinghe et al. (2011), the authors present structural constraint-aware VM placement to improve the performance and availability of services deployed in Infrastructure as a Service (IaaS) clouds. Their approach focuses on creating initial VM placement to satisfy three types of structural constraints: (i) demand constraints which define lower bounds on resource allocations that each VM requires, (ii) availability constraints which describe the collocation and anti-collocation constraints on VM placement, and (iii) communication constraints which describe the communication costs between pairs of VMs. Their approach models a data center as a tree structure based on the physical network topology and the logical groupings of the data center. Similarly, our approach can be extended to include the hierarchical structure of a data center. In addition, our approach focuses on the dynamic placement of VMs while minimizing the migration overhead.

The authors of Khanna et al. (2006) develop an algorithm for migrating VMs when performance problems are detected. In their approach, they assume a mapping of SLA to VM resource utilization. When the thresholds on the resource utilization are exceeded, the VM re-allocation procedure is triggered. Their approach tries to minimize the number of VM migrations. Our approach is application-independent and does not monitor the performance of the individual VMs.

Server consolidation problem is formally analyzed in the context of multi-dimensional bin packing problem in Speitkamp and Bichler (2010). Several heuristics are compared, including FFD and linear-programing relaxation based heuristics. In Gao et al. (2012), an ant colony optimization metaheuristic is used to solve the server consolidation problem. In Deng et al. (2013), the authors propose a new server consolidation manager which uses linear programing to find optimal configurations and then uses an optimized topological-sorting-based migration order generation method to reduce the reconfiguration costs. This is achieved by overlapping the migration processes of different VM when possible. All of these techniques do not consider the VM data migration overhead incurred in the reconfiguration plan.

The work of Grama and Kumar (1999) surveys several parallel algorithms used to solve discrete optimization problems such as the server consolidation problem. A discrete optimization problem is often formulated as the problem of finding a path in a graph (the state space graph) from a designated initial node to one of several possible final nodes. The authors also review several techniques to search the state space such as varieties to depth-first and best-first search and discusses how these algorithms can be parallelized. CPN Tools includes limited functionality to control how the state space is generated. However, in order to scale our approach to larger clusters, it is of interest to explore the use of these techniques in the context of our CPN model.

4. The proposed approach

To illustrate our approach, we develop a CPN model for the configuration in Fig. 1. A comprehensive guide to Colored Petri Nets and CPN Tools, including formal definitions, can be found in Jensen and Kristensen (2009). Section 4.1 describes the CPN model and Section 4.2 describes the generation and analysis of the state space using CPN Tools.

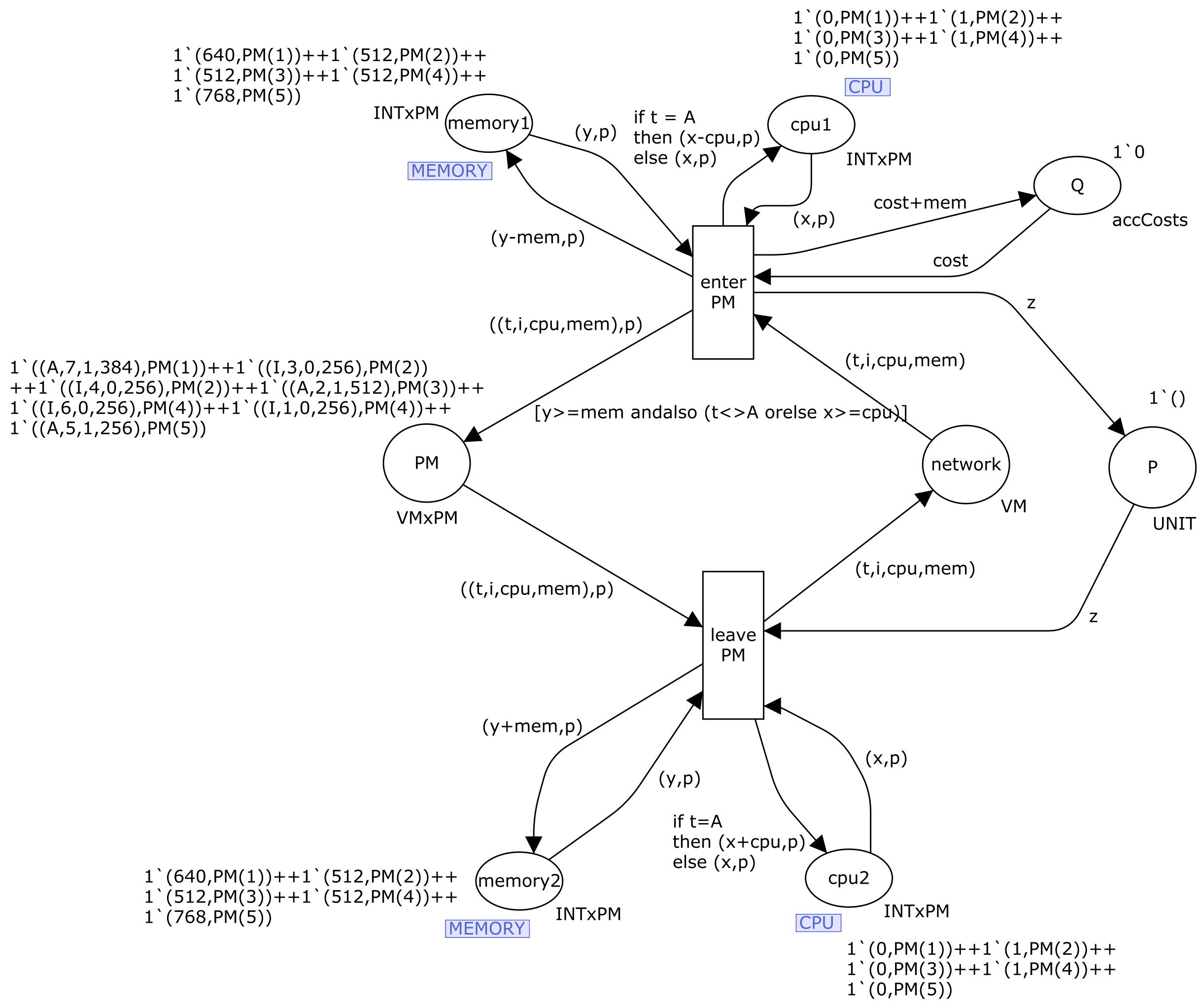
4.1. The CPN model

The CPN model is shown in Fig. 3. In this model, tokens in place *PM* specify the placement of VMs on the PMs. When a VM is migrated between two PMs, the corresponding token moves via the place *network*. Fusion sets *CPU* and *MEMORY* are used to record the available CPU and memory resources respectively on each PM as VMs are continuously migrated. Place *Q* records the total accumulated migration cost.

The color sets are defined as follows:

```
colset UNIT = unit;
colset INT = int;
colset BOOL = bool;
colset STRING = string;
colset vmStatus = with A | I;
colset vmID = int;
colset cpuReqt = int;
colset memReqt = int;
colset accCosts = int;
colset VM = product vmStatus * vmID * cpuReqt *
memReqt;
colset PM = index PM with 1..numMachines;
colset INTxPM = product INT * PM;
colset VMxPM = product VM * PM;
```

The variables are declared as follows:



1`(640,PM(1))++1`(512,PM(2))++ 1`(512,PM(3))++1`(512,PM(4))++ 1`(768,PM(5))
INTxPM
memory1
MEMORY
(y,p)
if t = A then (x-cpu,p) else (x,p)
1`(0,PM(1))++1`(1,PM(2))++ 1`(0,PM(3))++1`(1,PM(4))++ 1`(0,PM(5))
CPU
cpu1
INTxPM
(x,p)
cost+mem
Q
1`0
accCosts
cost
(y-mem,p)
enter PM
z
((t,i,cpu,mem),p)
(t,i,cpu,mem)
1`((A,7,1,384),PM(1))++1`((I,3,0,256),PM(2)) ++1`((I,4,0,256),PM(2))++1`((A,2,1,512),PM(3))++ 1`((I,6,0,256),PM(4))++1`((I,1,0,256),PM(4))++ 1`((A,5,1,256),PM(5))
[y>=mem andalso (t<>A orelse x>=cpu)]
1`()
PM
VMxPM
network
VM
P
UNIT
((t,i,cpu,mem),p)
(t,i,cpu,mem)
z
leave PM
(y+mem,p)
(y,p)
(x,p)
if t=A then (x+cpu,p) else (x,p)
1`(640,PM(1))++1`(512,PM(2))++ 1`(512,PM(3))++1`(512,PM(4))++ 1`(768,PM(5))
memory2
MEMORY
INTxPM
cpu2
CPU
INTxPM
1`(0,PM(1))++1`(1,PM(2))++ 1`(0,PM(3))++1`(1,PM(4))++ 1`(0,PM(5))

Fig. 3 The CPN model for the configuration in Fig. 1.

```
val numMachines = 5;
var x,y : INT;
var t: vmStatus;
var i: vmID;
var cpu: cpuReqt;
var mem: memReqt;
var cost:accCosts;
var z: UNIT;
var v: VM;
var p:PM;
```

The color set *VM* is used to model a virtual machine. A VM has a status (active or inactive), an identification number, a number which represents how many processing units are required by the VM, and a number which represents its memory size. For example, the token color *(A, 2, 3, 512)* represents the virtual machine VM2 which is active. *VM2* requires three processing units and its memory size is 512 MB.

To model the available processing units in a given PM, we use the fusion set *CPU*. The places *cpu1* and *cpu2* are members of the fusion set. Places that are in a fusion set always share the same marking. Tokens of color type *INTxPM* reside in the compound place represented by the fusion places of *CPU*. The value of a token represents the number of free processing units in the corresponding PM. Thus, in the initial marking of Fig. 3, one token in the fusion place *cpu1* has color *(0,PM(1))* since initially *PM1* holds an active VM (*VM7*) and thus *PM1* has no free processing units. An active VM cannot enter a PM if the number of free processing units in the PM is less than that is required by the VM (see the guard of the transition *enter PM*). If the VM enters the PM, the value of the token in the compound place of *CPU* is updated (see the expression of the arc from transition *enter PM* to place *cpu1*). This is analogous to the case when a VM leaves the PM.

The fusion set *MEMORY* is used to model the available memory in a given PM. The places *memory1* and *memory2* are members of the fusion set. Tokens of color type *INTxPM* resides in the compound place represented by the fusion places of *MEMORY*. The value of a token represents the amount of free memory in the corresponding PM. Thus, in the initial marking of Fig. 3, one token in the fusion place *memory1* has color *(640,PM(1))* since *PM(1)* initially hosts *VM7* and thus the free amount of memory in *PM(1)* is $1024 - 384 = 640$ MB. As the guard of the transition *enter PM* shows, a VM cannot enter a PM if the memory size of the VM exceeds the amount of free memory in the PM. When a VM enters a PM, the value of the token in the compound place of *MEMORY* is updated (see the expression of the arc from transition *enter PM* to place *memory1*). This is analogous to the case when a VM leaves the PM.

A technique we employ to reduce the state space is to enforce that a single token can reside in the place *network* at any given marking. This is accomplished by using the place *P*. This is justified because the objective is to find an optimal reconfiguration plan which minimizes the total VM migration overhead. Each migrated VM incurs migration overhead that is equal to its memory size and thus migrating two or more VMs in parallel does not affect the migration overhead. This significantly reduces the state space without impacting the resulting optimal reconfiguration plan.

One approach to include placement constraints in the CPN model is to add the corresponding conditions to the guard of the transition *enter PM*. For example, suppose that *VM5* must not be consolidated on *PM1* for the configuration in Fig. 1. We can add the following conjunct to the the guard of the transition *enter PM*: *not (p = PM(1) andalso i = 5)*. We repeat the same for every placement constraint, thus disabling the corresponding bindings.

4.2. State space generation and analysis

We use the state space tool of CPN Tools to find an optimal reconfiguration plan. Fig. 4 shows the query functions used to generate and search through the state space. These queries are written in the CPN ML programing language (presented in Chapter 3 in Jensen and Kristensen, 2009). For a given marking represented by *n*, the function *numIdleMachines* returns the number of physical machines which are idle. An idle PM does not host any VM and thus there is no token in place *PM* corresponding to the PM. The function *numBusyMachines* returns the number of busy (nonidle) machines and hence it equals *numMachines – numIdleMachines(n)*. The variable *numMachines* is declared earlier with the value 5.

Block 1 computes the minimum number of nonidle machines that is necessary to consolidate the VMs. This is done by defining the predicate *DesiredTerminal1* which returns true if the marking represented by *n* satisfies the condition that there is no *VM* token in place *network*. To find the minimum number of busy (nonidle) machines, we use the CPN ML predefined function *SearchNodes*. *SearchNodes* applies the combination function *Int.min* which takes two integer arguments and returns their minimum. The *SearchNodes* function only explores those markings which satisfy *DesiredTerminal1*. The output of the function is the minimum number of nonidle machines necessary in any marking satisfying *DesiredTerminal1*. It is stored in the variable *x* which will be used in the next block.

Block 2 searches for the optimal reconfiguration plan. This is done by searching for the marking corresponding to a configuration which uses the minimum number of nonidle machines *x* and which minimizes the total accumulated migration cost. The predicate *DesiredTerminal2* returns true if the marking represented by *n* satisfies *DesiredTerminal1* and that the number of nonidle machines is equal to *x*. The function *tot_cost* returns the total accumulated migration cost for a given marking which is equal to the value of the token residing in place *Q*.

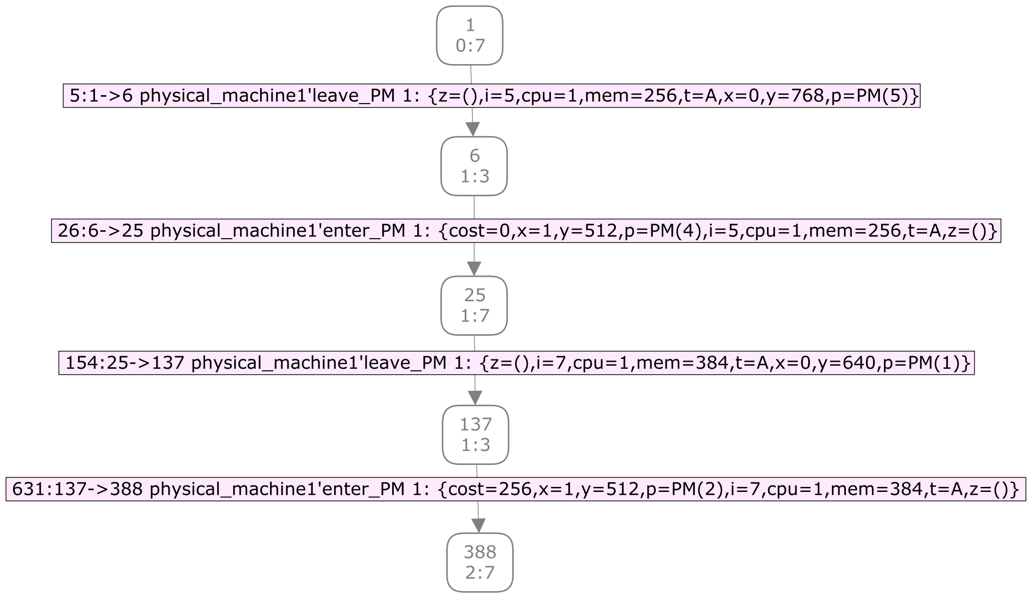
To find a minimal configuration, we use the function *SearchNodes* twice. In the first time, we use it to find the minimum value for the total accumulated migration cost over all markings which satisfy *DesiredTerminal2* (this value is stored in *mc*). In the second time, we use it to find the markings which satisfy *DesiredTerminal2* and whose total accumulated migration cost is equal to *mc*. The output of the second *SearchNodes* is the list of all markings corresponding to the minimal configurations. Starting from this marking, using CPN Tools the state space can be traversed backward to find a path from the initial marking. This path represents an optimal reconfiguration plan. Fig. 5 shows an optimal reconfiguration plan which corresponds to the optimal reconfiguration plan discussed in Section 2.

```
val max_tot_cost: int = 640;
val max_Secs: Int32.int = 60;
fun idleMachine (n,p:PM)= case List.exists (fn v => #2(v)=p) (Mark.physical_machine1'PM 1 n) of
                                true => 0 |
                                false=> 1;

fun numIdleMachines n=
  List.foldl
(fn (p:PM,x:int) => x+(idleMachine(n,p) )) 0 (PM.all());

fun numBusyMachines n = numMachines-numIdleMachines(n);

fun tot_cost n =
let
val accCostsToken = Mark.physical_machine1'Q 1 n;
in
hd(accCostsToken)
end;

(* Set Occ options and Calculate OCC *)
OGSet.BranchingOptions{TransInsts = NoLimit,
           Bindings = NoLimit, Predicate = fn n => (tot_cost(n) <= max_tot_cost)};
OGSet.StopOptions{Nodes=NoLimit, Arcs=NoLimit,Secs=max_Secs, Predicate = fn _ => false};
CalculateOccGraph();


(* Block 1 *)
(* Find the minimum number of necessary machines with the desired terminal*)
fun DesiredTerminal1 n= (Mark.physical_machine1'network 1 n == empty);
val x=SearchNodes(EntireGraph, DesiredTerminal, NoLimit, numBusyMachines, numMachines, Int.min);


(* Block 2 *)
(* Find desired state with minimum number of machines  and minimum total accumulated costs *)
fun DesiredTerminal2 n= DesiredTerminal1(n) andalso numBusyMachines(n) = x;
val mc = SearchNodes(EntireGraph, DesiredTerminal2, NoLimit, tot_cost, max_tot_cost, Int.min);
fun hasMinCost n = DesiredTerminal2(n) andalso tot_cost(n)=mc;
SearchNodes(EntireGraph, hasMinCost, NoLimit,fn n => n,[],op ::);
```

Fig. 4 The CPN ML queries used to generate and search through the state space for the CPN model of Fig. 3.

In generating the state space, it is necessary to set the branching options to specify the conditions under which the successors of a node are not calculated. Since the total accumulated migration cost can grow indefinitely as they are migrated, it is important to provide an upper bound for the total accumulated cost. This can be done in the state space tool of CPN Tools using the *OGSet.BranchingOptions* function as follows:

```
OGSet.BranchingOptions
TransInsts = NoLimit, Bindings = NoLimit,
Predicate = fn n => (tot_cost(n) <=
max_tot_cost)};
```



```
1
0:7
5:1->6 physical_machine1'leave_PM 1: {z=(),i=5,cpu=1,mem=256,t=A,x=0,y=768,p=PM(5)}
6
1:3
26:6->25 physical_machine1'enter_PM 1: {cost=0,x=1,y=512,p=PM(4),i=5,cpu=1,mem=256,t=A,z=()}
25
1:7
154:25->137 physical_machine1'leave_PM 1: {z=(),i=7,cpu=1,mem=384,t=A,x=0,y=640,p=PM(1)}
137
1:3
631:137->388 physical_machine1'enter_PM 1: {cost=256,x=1,y=512,p=PM(2),i=7,cpu=1,mem=384,t=A,z=()}
388
2:7
```

Fig. 5 A path in the state space representing an optimal reconfiguration plan for the configuration in Fig. 1.

This ensures that a partial state space is generated in which the total accumulated migration cost does not exceed *max_tot_cost* which is declared at the beginning of the CPN ML queries. In addition to using the branching options, it is possible to set the stop options in the state space tool to determine when the calculation of the state space stops. For example, the following function stops calculating the state space after *max_Secs* number of seconds (also declared at the beginning of the CPN ML queries):

```
OGSet.StopOptions{Nodes=NoLimit,
Arcs=NoLimit,Secs=max_Secs, Predicate = fn _ =>
false};
```

Calculating the state space can then be continued. This pattern of stopping and continuing state space calculation can be repeated noting that the optimal reconfiguration plans that are discovered at the end of each calculation round are the optimal plans computed so far. This is an advantage property for our approach that can exploited to deal with potentially large state spaces.

5. Evaluation

The presented approach in Section 4 may suffer from the state explosion problem. For larger clusters than the one discussed in Section 2, the state space can have a very large number of reachable states making it infeasible to find an optimal reconfiguration plan. This section presents the results of applying several techniques to alleviate the state explosion problem. The techniques do not guarantee an optimal reconfiguration plan; rather, they find good reconfiguration plans with better results than FFD in reasonable time and computing power.

5.1. Controlling the parameters max_tot_cost and max_Secs

Consider the CPN ML queries used to generate and analyze the state space of the CPN model (see Fig. 4). There are two variables that limit the size of the explored state space: *max_tot_cost* and *max_Secs*. The first variable places an upper bound on the total accumulated migration cost while the second variable allows CPN Tools to stop generating the state space after the specified period of time in seconds. To be comparable to other server consolidation approaches, we set *max_Secs* to 60 s (see Murtazaev and Oh (2011)). Thus, the only variable that can be set to control the size of the state space is *max_tot_cost*.

The first technique to deal with the state explosion problem is to set the variable *max_tot_cost* to smaller values. Table 1 shows information on the size of the state spaces computed under two different settings. These settings use different upper bounds on the total accumulated migration cost (i.e., different values for *max_tot_cost*). As the results indicate for the second setting, using a lower value for *max_tot_cost* significantly cuts down the size of the state space and the time that is needed to generate the state space. Note that the state space tool was used on a Dell desktop computer equipped with a 3.00 GHz dual-core processor and 2 GB RAM.

This technique suffers from two main limitations. First, determining a suitable value for *max_tot_cost* may not be trivial. It can be based on the use of other heuristics (such as FFD), however the size of the generated state space is very sensitive to the chosen value. Second, as discussed next, the state space exponentially increases in size when considering larger cluster sizes. In such cases, the state space generated in 60 s explores states (nodes) with total accumulated migration costs much less than *max_tot_cost* and the returned reconfiguration plans are far from optimal in terms of minimizing the number of physical machines used. Hence, setting a suitable value for *max_tot_cost* has less impact for such cases.

To confirm the second limitation discussed in the previous paragraph, Table 2 shows the results of applying our approach on a cluster of 10 PMs and 14 VMs. We assume that the cluster is built by composing two instances of the five-machine cluster having the initial configuration of Fig. 1 while renaming the PMs and VMs such that each of them gets a unique name. This is chosen since the results in Section 4 provide hints on what to expect in terms of the minimum number of physical machines and the minimum accumulated migration cost in an optimal reconfiguration plan. We refer to this setup as a cluster built using a multiple of two instances of the basic configuration of Fig. 1. Later, we will use higher multiples to show the results on even larger clusters. The results in Table 2 assumes that *max_Secs* is set to 60 s.

There are two observations one can make on the results presented in Table 2. First, the obtained reconfiguration plan is not optimal since there exists another plan which can reach a configuration using 6 PMs rather than 8 PMs. This can be achieved using the same steps of the optimal reconfiguration plan for the five-machine cluster having the basic configuration of Fig. 1 twice: one for each composite cluster. Second, it is clear that setting *max_tot_cost* to larger values has no impact on the results. This is because there is a maximum limit on the state space that can be generated in 60 s. Note the similarity between the resulting number of nodes and arcs in the two settings. The same observation can be made when using higher multiples, however the obtained reconfiguration plan becomes worst in terms of minimizing the number of PMs. For example, when using a multiple of three instances of the basic configuration of Fig. 1 and setting the Upper Bound to 2000 MB, the minimum number of PMs is found to be 13 PMs assuming *max_Secs* is set to 60 s. Another reconfiguration plan exists which can reach a configuration with 9 PMs.

5.2. Controlling the parameter Bindings

To deal with the problems discussed in Section 5.1, we propose a new technique that exploits a feature in CPN Tools which allows to control how the state space is generated. One of

Table 1 State space statistics under different upper bounds on the total accumulated migration cost. In the last column, the total cost is for a computed optimal reconfiguration plan.

Setting	Upper bound	State space statistics			Total accumulated migration cost
		#Nodes	#Arcs	#Seconds	
1	2000 MB	14,855	30,617	60	640
2	640 MB	3128	6300	3	640

Table 2 State space statistics for a cluster built using a multiple of two instances of the basic configuration of Fig. 1.

Setting	Upper bound	State space statistics			Minimum number of PMs	Total accumulated
		#Nodes	#Arcs	#Seconds		migration cost
1	2000 MB	21,711	29,456	60	8	512
2	640 MB	20,209	27,944	60	8	512

Table 3 A comparison between the results of using the CPN Approach using *Bindings* = *3* and FFD.

	CPN approach		FFD	
Multiples	Minimum number of PMs	Total accumulated migration cost	Minimum number of PMs	Total accumulated migration cost
2	7	1024	7	2432
3	11	1280	11	3712
4	16	1152	16	2816
5	21	1536	21	2944
6	26	1152	26	3072

the arguments to the ML function *OGSet.BranchingOptions* in Fig. 4 is *Bindings* which specifies the maximal number of enabled bindings to be used to find successor markings for each node in the state space. By setting *Bindings* to a specific number (the default is *NoLimit*) the generation of the state space becomes more depth-first rather than the default bread-first fashion of state space generation in CPN Tools. We set *Bindings* to three since this induces enough randomness to improve the discovered optimal reconfiguration plans. At the same time, our experiments indicate that setting *Bindings* to three achieves good results given the 60 s restriction on the period of time to generate the state space. Table 3 shows the results of applying the technique on clusters built using multiples of two to six instances of the basic configuration of Fig. 1 assuming *max_Secs* is set to 60 s and that the Upper Bound is set to 640 MB × *the number of multiples* in order to accommodate the different cluster sizes. The table also includes the results of executing FFD until reaching a configuration using the same minimum number of PMs.

5.3. Other methods

Several methods have been proposed to deal with the state explosion problem for CPN models. These include the stubborn-set method (Valmari, 1988), the sweep-line method (Christensen et al., 2001), and the equivalence method (presented in Section 8.4 in Jensen and Kristensen (2009)). These methods can be applied on certain cluster configurations in which certain dependencies and symmetries are exploited. However, generally speaking, these methods may not apply on cluster configurations with a large degree of heterogeneity. For this reason, we do not explore such methods. Besides, CPN Tools currently does not implement any state reduction method.

6. Conclusion

This paper has presented a new approach to server consolidation using CPNs and CPN Tools. Our approach takes into consideration the total data migration overhead which has not been considered in other approaches. The approach suggests several techniques to cope with the resulting large state spaces. One of the added advantages using our approach is the property that generating the state space can be paused at any time returning the best result computed so far. This increases the reactivity of our approach and thus its applicability to larger clusters. Another strength of our approach is the inherent flexibility of the CPN models to capture several constraints on the placement of the VMs. The main challenge is to efficiently deal with the resulting exponentially increasing state spaces. As a future work we plan to explore other techniques for state space analysis aiming to improve the scalability of our approach to work on larger clusters typically used in common data centers.

Acknowledgments

This research was supported by the Research Center (RC) of the College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia (project ID: RC1303103).

References

Bobroff, N., Kochut, A., Beaty, K.A., 2007. Dynamic placement of virtual machines for managing SLA violations. In Proceedings of the Symposium on Integrated Network Management, pp. 119–128.

Caprara, A., Toth, P., 2001. Lower bounds and algorithms for the 2-dimensional vector packing problem. Discrete Appl. Math. 111 (3), 231–262.

Christensen, S., Kristensen, L.M., Mailund, T., 2001. A sweep-line method for state space exploration. In Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems, pp. 450–464.

Clark, C., Fraser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I., Warfield, A., 2005. Live migration of virtual machines. In Proceedings of the Symposium on Networked Systems Design and Implementation, pp. 273–286.

Coffman, E.G., Garey, M.R., Johnson, D.S., 1997. Approximation algorithms for bin packing: a survey. In: Approximation Algorithms for NP-hard Problems. PWS Publishing Co, pp. 46–93.

CPLEX. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.

CPN Tools. http://cpntools.org/.

Deng, L., Jin, H., Wu, S., 2013. Merger: server consolidation in virtualized environment. In Proceedings of the International Conference on Dependable, Autonomic and Secure Computing, pp. 606–612.

Ferreto, T.C., Netto, M.A.S., Calheiros, R.N., De Rose, C.A.F., 2011. Server consolidation with migration control for virtualized data centers. Future Gener. Comput. Syst. 27 (8), 1027–1034.

Gao, Y., Guan, H., Qi, Z., Wang, B., 2012. An ant colony system algorithm for the problem of server consolidation in virtualized data centers. J. Comput. Inform. Syst. 8 (16), 6631–6640.

Grama, A., Kumar, V., 1999. State of the art in parallel search techniques for discrete optimization problems. IEEE Trans. Knowl. Data Eng. 11 (1), 28–35.

Hermenier, F., Lorca, X., Menaud, J.-M., Muller, G., Lawall, J., 2009. Entropy: a consolidation manager for clusters. In Proceedings of the Conference on Virtual Execution Environments, pp. 41–50.

Jayasinghe, D., Pu, C., Eilam, T., Steinder, M., Whally, I., Snible, E., 2011. Improving performance and availability of services hosted on IaaS clouds with structural constraint-aware virtual machine placement. In Proceedings of the Conference on Services Computing, pp. 72–79.

Jensen, K., Kristensen, L.M., 2009. In: Coloured Petri Nets – Modelling and Validation of Concurrent Systems. Springer.

Khanna, G., Beaty, K., Kar, G., Kochut, A., 2006. Application performance management in virtualized server environments. In Proceedings of the Network Operations and Management Symposium, pp. 373–381.

Mehta, S., Neogi, A., 2008. ReCon: a tool to recommend dynamic server consolidation in multi-cluster data centers. In Proceedings of the Network Operations and Management Symposium, pp. 363–370.

Meng, X., Pappas, V., Zhang, L., 2010. Improving the scalability of data center networks with traffic-aware virtual machine placement. In Proceedings of the Conference on Information Communications, pp. 1154–1162.

Murtazaev, A., Oh, S., 2011. Sercon: server consolidation algorithm using live migration of virtual machines for green computing. IETE Tech. Rev. 28 (3), 212–231.

Shrivastava, V., Zerfos, P., Lee, K.-W., Jamjoom, H., Liu, Y.-H., Banerjee, S., 2011. Application-aware virtual machine migration in data centers. In Proceedings of the Conference on Information Communications, pp. 66–70.

Skiena, S., 2008, second ed.. In: The Algorithm Design Manual Springer.

Speitkamp, B., Bichler, M., 2010. A mathematical programming approach for server consolidation problems in virtualized data centers. IEEE Trans. Serv. Comput. 3 (4), 266–278.

Spieksma, F.C.R., 1994. A branch-and-bound algorithm for the two-dimensional vector packing problem. Comput. Oper. Res. 21 (1), 19–25.

Valmari, A., 1988. Error detection by reduced reachability graph generation. In Proceedings of the European Workshop on Application and Theory of Petri Nets, pp. 95–112.

Wood, T., Shenoy, P., Venkataramani, A., Yousif, M., 2007. Black-box and gray-box strategies for virtual machine migration. In Proceedings of the USENIX Symposium on Networked Systems Design and Implementation.