An Integrated Business Rules and Constraints Approach to Data Centre Capacity Management

Roman van der Krogt, Jacob Feldman, James Little and David Stynes

Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland

Abstract. A recurring problem in data centres is that the constantly changing workload is not proportionally distributed over the available servers. Some resources may lay idle while others are pushed to the limits of their capacity. This in turn leads to decreased response times on the overloaded servers, a situation that the data centre provider wants to prevent. To solve this problem, an administrator may move (reallocate) applications or even entire virtual servers around in order to spread the load. Since there is a cost associated with moving applications (in the form of down time during the move, for example), we are interested in solutions with minimal changes. This paper describes a hybrid approach to solving such resource reallocation problems in data centres, where two technologies have to work closely together to solve this problem in an efficient manner.

The first technology is a Business Rules Management System (BRMS), which is used to identify which systems are considered to be overloaded on a systematic basis. Data centres use complex rules to track the behaviour of the servers over time, in order to properly identify overloads. Representing these tracking conditions is what the BRMS is good for. It defines the relationships (business constraints) over time between different applications, processes and required resources that are specific to the data centre. As such, it also allows a high degree of customisation.

Having identified which servers require reallocation of their processes, the BRMS then automatically creates an optimisation model solved with a Constraint Programming (CP) approach. A CP solver finds a feasible or the optimal solution to this CSP, which is used to provide recommendations on which workload should be moved and whereto. Notice that our use of a hybrid approach is a requirement, not a feature: employing only rules we would not be able to compute an optimal solution; using only CP we would not be able to specify the complex identification rules without hard-coding them into the program. Moreover, the dedicated rule engine allows us to process the large amounts of data rapidly.

1 Introduction

Data centres are *"buildings where multiple servers and communication gear are colocated because of their common environmental requirements and physical security needs, and for ease of maintenance"* [1]. They have become increasingly more important due to two trends. Firstly, there is a need to process larger and larger workloads and store ever increasing amounts of data. An obvious example of this would be search engines on the World Wide Web, but also consider supermarkets that use loyalty cards to track what their customers buy. More companies storing and analysing larger amounts of data means that an increased amount of computing power is needed to satisfy the demand. At the same time, computing and storage is moving from PCs to internet services and so more centralised storage and computing power is required. Besides reasons of scale and ease of management (such as centralised backups), there is also a large economical reason: applications run at a lower cost per user, as servers may be shared by many active users and many more inactive ones. Virtualisation is the latest trend in this area. One can now acquire virtual servers, that scale with the computing resources required (*cloud computing*). Many such virtual servers can reside on the same physical server, providing low cost solutions.

However, data centres also come with new computational challenges. The problem we will focus on in this paper is due to the dynamic nature of the environment. Most processes behave in a very erratic way, switching from using few resources to many and back.[1] For example, consider a mail server. This remains dormant until email arrives at which point it wakes up and processes the email. One can have many dormant processes on a server, but only a limited number of active ones. When too many are active, the response times decrease and the service level goes down. This results in the appearance of an application running slow and hence affects the user experience.

Similar concerns arise in cloud computing. Many small virtual servers can be hosted on a single physical server, but when too many want to scale up simultaneously (whether in terms of CPU capacity, available memory, bandwidth or otherwise) there is simply not enough room to accommodate all of them. Such situations are currently dealt with through human operators. From their control room, they are able to see the characteristics of the servers (including the current CPU load, memory and bandwidth consumption, and possibly including factors such as power usage and temperature). When they recognise unwanted patterns (e.g. a server being overloaded for a certain period) they will investigate and manually solve the problem by moving processes around. The current state-of-the-art support tools for data centre administrators (as, e.g. the CapacityAnalyzer product of VKernel [2]) provide them not only with a monitoring tool, but can also diagnose or even predict problems and bottlenecks, i.e. indicate which servers are running critical . However, the tools do not provide a recommendation as to which process should be moved, or whereto. The system we describe in this paper is a comprehensive system, that

1. Acts as an early warning system for potential problems with the servers in the data centre; and

[1] For convenience, we will use the term "process" to refer to the particular objects that we can reallocate between servers. These could be processes in the traditional sense, but in different circumstances, it could mean a complete virtual server consisting of many traditional processes.

2. Proposes solutions in the event problems are indeed detected.

The problem naturally decomposes into two separate subproblems of *(i)* problem identification and formulation and *(ii)* problem resolution. Correspondingly, the proposed system uses two different solution techniques, each most suited to the particular subproblem. We will use a Business Rule Management System (BRMS) to analyse the information stream that is provided by the monitoring software. To this end, we define in the BRMS the relationships between the different applications, processes and required resources. Using this information, the BRMS can efficiently track the state of the servers over time. By applying business rules, it identifies resource overloading situations and potential bottlenecks. If there are any issues detected, we will formulate an appropriate resource reallocation problem to be solved using Constraint Programming (CP). The purpose of the CP formulation is to find a set of minimal changes to the current configuration in order to resolve the identified problem. (For example, in the case of a CPU overload, a busy application could be moved from the server to one with a low level of CPU utilisation.) As different types of data centres prefer different kinds of reallocations, the user can specify their objectives through the BRMS and have them taken into account when generating the CSP.

A key contribution of our work is the fact that the CP formulation is automatically derived through the rule engine. It puts the business specialists (i.e. data centre operators) in charge of the workload re-configuration process allowing them to concentrate on what they want to achieve, instead of how this should be achieved. Existing combinations of BRMS and optimisation (which we will discuss in a later section) have used rules to provide the input to the optimisation problem, and/or use rules to direct the search. In contrast, the entire problem formulation is driven by the rules in our approach. The rules that are triggered will add variables and constraints to the model, thus constructing it in real-time, based on the actual, data centre-specific situation. A second contribution is derived from the integration of CP within Business Rules, which allows people with no expert knowledge of optimisation in general, or CP in particular, to express and change their problem simply through the rule editor. This makes it easier for business analysts (to whom the modern business rules systems are oriented) to maintain CP models.

The remainder of the paper is organised as follows. First, we provide a formal problem specification. Then, in Section 3, we detail the proposed solution. Section 4 provides an experimental analysis of our approach. After discussing related work, we draw conclusions in Section 6 and provide pointers for future work.

2 Formal Problem Specification

Our data centre consists of a set $S = \langle s_{1,} \dots, s_n \rangle$ of n servers. Depending on the type of data centre, these servers can be used to provide computing power, data storage, a combination of these, or any other purpose. This is of no consequence to the model itself. Furthermore, there is a set $P = \langle p_1, \dots, p_m \rangle$ of processes. At regular time intervals, a set of l sensors provide information on the processes, e.g. their CPU load, the amount of memory required or energy consumption. A function $\sigma_i^t : P \to \mathbb{R} \cup \{\perp\}$, $i \in [0, l]$ provides the output of a sensor for a given time point and a given process, i.e. $\sigma_i^t(p)$ equals the value measured by sensor i at time t for the process p. If a process is not active at a certain time point t (e.g. because it has finished), the output of σ_i^t, $i \in [0, l]$ is undefined, i.e. $\sigma_i^t = \perp$. By default, sensor 0 returns the server that the process is running on, i.e. $\sigma_o^t(p) = j$ iff p runs on s_j at time t.

The total requirements placed on a server s_j can be computed from the processes running on that server as follows:

$$\sigma_i^t(s_j) = \sum_{\{p \mid \sigma_0^t(p)=j\}} \sigma_i^t(p) \quad i \in [1, l]$$

We will use $\sigma^t(s) = \langle \sigma_0^t(s), \ldots, \sigma_l^t(s) \rangle$ to denote the set of values of all sensors for a given server at a particular time. We let Σ denote the set of all possible combinations of sensor readings, i.e. $\sigma^t(s) \in \Sigma$.

To identify which servers are of interest, we introduce a classification $\mathbb{S}$ of possible labels (for example, this could be equal to $\mathbb{S} = \{\text{critical, high, medium, low}\}$). We assume there is a function $\Delta : S \times \Sigma^z \to \mathbb{S}$ that, given a set of sensor readings for the past z time steps, can give the state a server is in. In order to optimise our solution, we introduce a cost function $\mathit{cost} : \mathbb{S} \to \mathbb{R}^+$ that, given a label, returns a virtual cost value for that particular label.

Through the classification of servers, we may identify that an unwanted situation has arisen. For example, we may find a number of servers that are classified as critical. To address this issue, we could move processes away from these servers onto others. Thus, a solution is a reallocation of processes that brings the data centre to an improved state. In terms of the model, we want to compute a set $\{\sigma_o^{t+1}(p) \mid p \in P\}$.

There are a number of constraints that should hold for a solution. Firstly, there is a maximum value max_i that we want to satisfy for each sensor $\sigma_i^t(s)$ of a server s. Secondly, some processes may never run together, i.e. be present on the same server. To this end, there is a function $\mathit{incompatible} : P \to 2^P$ that given some process returns the list of all processes that cannot share the same resources with that process.[2] Finally, some processes may need a particular set of servers. The function $\mathit{possible} : P \to 2^S$ denotes the possible servers that a process can run on. (Thus, $\mathit{possible}(p) = S$ if there are no restrictions on p.)

The last aspect we introduce is a distance metric between servers. The function $\delta : S \times S \to \mathbb{R}^+$ returns the distance between two servers. This could simply be the physical distance, but is more likely to also take into account the network

[2] A possible reason for such an incompatibility constraint may be that a critical application is distributed over multiple servers to provide a level of fault tolerance. We should then prevent that two instances of this application are running on the same server, in order to guarantee the application is still available when the server crashes. Security aspects are another typical example.

topology in the data centre. When moving processes, we want to minimise the total distance of the processes that are moved. Note that, in general,

$$\min_{s \in \mathbb{S}} \mathit{cost}(s) > \max_{s_1, s_2 \in S} \delta(s1, s2) \tag{1}$$

That is, our first priority is in achieving acceptable levels for the states of the servers, with minimum movements of secondary importance.[3]

The full problem statement can now be formalised as follows.

$$\begin{array}{rl}
\textbf{Given} & S = \langle s_1, \ldots, s_n \rangle \text{ servers} \\
& P = \langle p_1, \ldots p_m \rangle \text{ processes} \\
& \sigma^0, \ldots \sigma^t \text{ sensor readings, where } \sigma^i = \langle \sigma^i_0, \ldots, \sigma^i_l \rangle \\
\textbf{Find} & \sigma^{t+1}_o(p) \text{ for each } p \in P \\
\textbf{Subject to} & \forall p \in P \cdot \sigma^{t+1}_i(p) = \sigma^t_i(p), i = [1, l] \\
& \forall s \in S \cdot \sigma^{t+1}_i(s) \leq \mathit{max}_i \\
& \forall p \in P \cdot \sigma^{t+1}_0(p) \in \mathit{possible}(p) \\
& \forall p_1, p_2 \in P \cdot p_1 \in \mathit{incompatible}(p_2) \implies \sigma^{t+1}_o(p_1) \neq \sigma^{t+1}_0(p_2) \\
\textbf{Minimising} & \sum_{s \in S} \mathit{cost}(\mathit{\Delta}(s, \sigma^{t+1}(s), \ldots \sigma^{t+1-z}(s))) \\
& \quad + \sum_{p \in P} \delta(\sigma^t_0(p), \sigma^{t+1}_0(p))
\end{array}$$

3 A Hybrid Approach

The problems naturally breaks down into three stages:

1. The identification of servers which are at the different risk levels;
2. The formulation of the re-distribution problem; and
3. Solving the generated problem to compute a new workload allocation with as few changes as possible.

3.1 Stage 1a: Rules-Based Problem Identification

The first stage relates to the implementation of the function $\mathit{\Delta}$ that we introduced in the previous section. The problem here is that this function may be very complex due to the many conditions over time that need to be taken into account. The biggest cause of this complexity is a desire for stability. It would be too costly to reallocate processes each time a certain sensor reading is too high and therefore, we need to establish a pattern emerging over time before we decide to take action. There is a also a second issue: flexibility. This is desired because not all variations of the problem are known ahead of time, and this

[3] However, note that $\mathit{cost}(s)$ might equal $\mathit{cost}(s) = 0$, in which case these states should be disregarded in Equation 1.

Table 1. A simple rule set

event	condition	action
ϵ	$\sigma_1 \leq 100$	**set** *label* = high
ϵ	$\sigma_1 \leq 90$	**set** *label* = medium
ϵ	$\sigma_1 \leq 60$	**set** *label* = low

allows a user to customise for their situation: the SLAs they have agreed with users, standard procedures, etc.

Most data centres already use monitoring systems that provide performance, resource utilisation and workload projection capabilities. The resource utilisation data usually contains a number of samples taken over monitored time intervals that range from 30 seconds to 30 minutes. This data indicates when and how frequently different utilisation spikes occur. Business rules are an ideal way to formulate the process of analysing the monitoring data and define spike frequencies and overload thresholds.

For the purpose of this research, we consider a business rule to be a triplet $b_\chi = \langle e, c, a \rangle$ [3], where χ is a list of variables, e describes an event that triggers the rule, c describes a condition over χ that has to be met, and a specifies the action to perform. Rules are grouped in *rule set*s. When an event happens, the rules in each applicable rule set $B = \{b_1, \ldots, b_r\}$ are evaluated in order, starting from b_1, and identifying the actual values with the variables χ. Depending on the strategy the BRMS employs, one or more of the matching rules is executed. Executing multiple matching rules allows more specific conditions to override generic ones. This is the strategy taken by our system.

By enumerating the different cases described by $\mathit{\Delta}$ we can capture the behaviour of $\mathit{\Delta}$ in a rule set. The trigger event for each rule is the update in the sensor readings; the condition describes the case to which it applies; and the action specifies the label that $\mathit{\Delta}$ outputs for that particular case. For example, consider the following very simple labeling function, with the number of timesteps equal to $z = 1$, $\mathbb{S} = \{\text{high}, \text{medium}, \text{low}\}$, and assuming $\sigma_1^t \in [0, 100]$ measures the CPU load in percentages:

$$\mathit{\Delta}(s, \sigma) = \begin{cases} \text{low} & \text{if } \sigma_1(s) \leq 60 \\ \text{medium} & \text{if } 60 < \sigma_1(s) \leq 90 \\ \text{high} & \text{if } 90 < \sigma_1(s) \leq 100 \end{cases}$$

This can be efficiently represented using the rule set of Table 1, where ϵ denotes an update to the sensor readings. More complicated functions can be represented by rule sets in a similar fashion; for example differentiating between different days of the week or time of day, or different kinds of processes. Given a proper interface, such as can be provided by Excel (cf. Figure 1 below), the rules can be defined and maintained by business analysts who understand the actual thresholds for different resource types for different time periods. Thus, the rules can be easily customised for different (types of) data centres.

Table 2. The previous rule set extended

event	condition	action	additional constraints
ϵ	$\sigma_1(s) \leq 100$	**set** *label* = high	**post** $\sigma_1^{t+1}(s) \leq 70$
ϵ	$\sigma_1(s) \leq 90$	**set** *label* = medium	
ϵ	$\sigma_1(s) \leq 60$	**set** *label* = low	

3.2 Stage 1b: Building the Optimisation Model

At the problem identification stage we can also start to build the features of the optimisation model in terms of the variables and constraints. Our solution provides the facility to describe constraints within the Business Rules environment and to add them to a CP model. For example, we can extend the rules in Table 1 with constraints on the utilisation of a server over the next period. Again, let σ_1 denote the CPU load, and assume that we want to constrain the CPU load for servers with a high load to at most 70%. This can be reflected in the business rules by including a statement that adds (posts) a new constraint, as shown in Table 2. These same rules, but now specified in our Excel interface, are shown in Figure 1. The developer view (the lightly coloured rows 4, 5 and 6) is normally hidden from the user's view, i.e. the business analyst would only see rows 7-10, and can manipulate the data there to reflect their situation (including the addition of rows to add more rules). The developer view shows how the rules relate to the business objects (e.g. specifying how the "CPU load is less than" condition is evaluated), and the effects on the business objects and the CSP. In this case, for example, the effect on the CSP is the creation of a new variable and the introduction of constraints on this variable, using the following Java snippet:

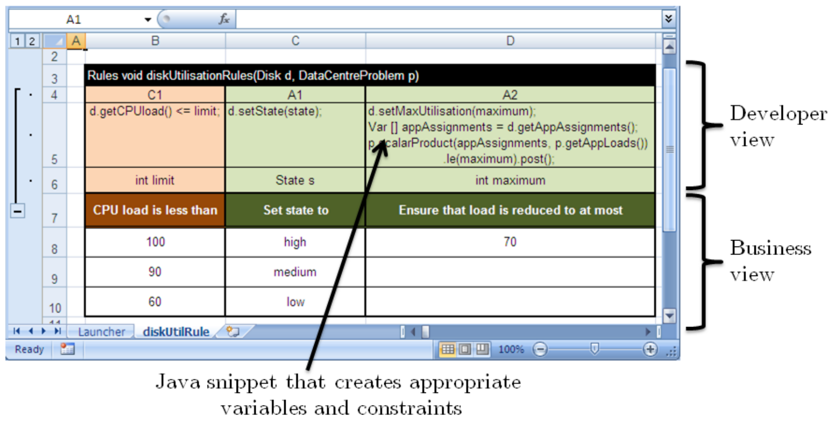
```
d.setMaxUtilisation(maximum);
Var [] appAssignments = d.getAppAssignments();
p.scalarProduct(appAssignments, p.getAppLoads()).le(maximum).post();
```

Notice also how the constraint variable is associated with the business object (through the `setBusinessObject()` method), so that other rules (or other applications of this rule) can find this variable, if necessary.

Besides the posting of simple constraints, we can introduce a certain amount of intelligence in the building of the model at this stage. As the number of servers can be very large, we believe that we should have the facility to limit the number of servers to consider in the solution. For example, we could limit ourselves to only those servers that have a high load, and those that have capacity to spare (e.g. those servers s for which $\sigma_1(s) \leq 30$). Again, the rules interface allows a data centre administrator to specify this efficiently.

3.3 Stage 2: Solving the Optimisation Model

With the number of servers totaling potentially a very high number, the problem may quickly become intractable. Therefore, we propose that a smaller model can be generated, depending on the size and nature of the problem defined by the


Developer view
Business view
Java snippet that creates appropriate variables and constraints
Launcher
diskUtilRule
Ready

Rules void diskUtilisationRules(Disk d, DataCentreProblem p)		
C1	A1	A2
d.getCPUload() <= limit;	d.setState(state);	d.setMaxUtilisation(maximum); Var [] appAssignments = d.getAppAssignments(); p.scalarProduct(appAssignments, p.getAppLoads()) .le(maximum).post();
int limit	State s	int maximum
CPU load is less than	Set state to	Ensure that load is reduced to at most
100	high	70
90	medium	
60	low	

Fig. 1. The rule set of Table 2 specified in Excel

rules. Our hypothesis is that a small problem will find better solutions within the available time than a larger model, using the same search strategy.

To explore this hypothesis, we propose two models: a global one, and a localised one. In the first (global) model, we look at the state of all servers, identify which servers require a reallocation of volumes and solve the resulting problem.

The second approach is described by the following two inter-leaving stages:

1. As each server is sequentially identified as high risk using the rules definition, a small local model is created to solve the problem, but within a restricted set of servers that have spare capacity available; followed by
2. An optimisation stage which generates a new workload allocation with as few changes as possible.

This approach is particularly relevant where the presence of high risk servers is fairly sparse and there is the opportunity to create several small problems.

4 Experimental Results

Following discussions with a large data centre hardware provider, we generated a set of benchmark problems to test our system with. The benchmark set focuses on the management of storage servers. Such servers do not run any applications, yet are dedicated to the storage and retrieval of information. Information is stored on virtual disks ("volumes") that may span many physical disks, even across different servers. Each volume has a list of applications that are using that particular volume, and a list of I/O operations per application. Different volumes may occupy the same physical disk. Thus, the total amount of data that is being transferred to/from a disk is determined by the activity of the different volumes on it. If several volumes with high transfer rates are stored on the same disk, the disk is overloaded and response times drop. In such a case, we want to

Table 3. Classification rules, $\sigma_1^1, \ldots, \sigma_1^z$ are the last z values of the load sensors; I is the interval between readings. For brevity, the event column is omitted, as is the Java code generating the model

condition	action
$\frac{\lVert\{\sigma_1^t \mid \sigma_1^t > 45\}\rVert}{z} \geq 40$	**set** *label* = medium
$I \geq 10 \wedge \exists i \in 1 \ldots z \cdot \sigma_1^i \geq 60$	**set** *label* = high
$I < 10 \wedge \exists i \in 1 \ldots z \cdot \sigma_1^i \geq 70$	**set** *label* = high
$I \leq 15 \wedge \exists i \in 1 \ldots z-2 \cdot \left[\sigma_1^i \geq 70 \wedge \sigma_1^{i+1} \geq 70 \wedge \sigma_1^{i+2} \geq 70\right]$	**set** *label* = very high
$I \geq 15 \wedge \exists i \in 1 \ldots z-1 \cdot \left[\sigma_1^i \geq 70 \wedge \sigma_1^{i+1} \geq 70\right]$	**set** *label* = very high
$I \geq 30 \wedge \exists i \in 1 \ldots z \cdot \left[\sigma_1^i \geq 70\right]$	**set** *label* = very high

redistribute the volumes over the active disks in such a way as to alleviate any bottle necks. Moving a volume is a costly operation, however, and so we want to minimise the number of moves over time.

Due to the nature of the problem, a number of additional constraints are present. These relate to the physical hardware that the systems employ. In particular, because of service level agreements that are in place, we can only move (part of) a volume from one physical disk to another, if these disks are of the same size, speed and type. Additionally, different parts of a RAID volume cannot share the same physical disk.[4]

Table 3 lists the rules that are used to classify the servers. Notice that the interval at which sensors are read differs between systems. Therefore, the time between sensor updates is included in the rules.

4.1 Description of the Benchmark Sets

Our benchmark set consists of 100 problem instances that were randomly generated from actual data. The problems have the following characteristics: we consider a storage array with n disks, each of which is represented by a server in our model. The array is used by $2n$ applications, where each application a_i requires space on between 1 and 3 disks (we say that the *virtual disk* for application a_i requires between 1 and 3 *volumes*). If application a_i requires, say, space on 2 disks, this gives rise to 2 processes in our model, $a_i^1$and a_i^2, that we have to accommodate. For security reasons, these cannot be allocated on the same disk (as they are part of the same RAID device). Therefore, an *incompatible* constraint is imposed over such sets of processes. Finally, only 4 applications can use a disk at the same time due to space restrictions. To this end, we introduce a sensor σ_{size} that reports $\sigma_{size}^t(p) = 25$ for each time step t and each process p. For any given server s, we require $\sigma_{size}(s) \leq 100$ for all time points. To reiterate,

[4] A RAID configuration (Redundant Array of Inexpensive Disks) allows data to be divided and replicated among multiple hard drives to increase data reliability and/or increase performance. These benefits are lost when the parts of the array ends up on the same disk.

the disks in the storage array are represented by the servers in our model; the volumes on those disks are represented by the processes running on the servers. We assume unit distance between all disks, in effect minimising the total number of changes we make to the allocation of applications to disks.

The values of the load sensors were chosen such that either $1/4^{th}$ or $1/8^{th}$ of the disks is considered to have a high or very high load, and the goal of the system is to achieve a maximum load of 60% for any given disk. As we specified the interval between sensor readings to be $I = 15$, this corresponds to ensuring that none of the disks will be classified as either high or very high when considering just the next time point. For each combination of the total number of disks and the number of overloaded ones, we generated 10 instances.

4.2 Models Considered

As indicated in Section 3.3, we consider two models. The first is a global model, in which the full set of disks is taken into account. We will refer to this as the "full" model. The other model, referred to as "iterative" considers each disk in turn, and if the rules indicate that this disk is classified as high or very high, we create a small CSP to solve the reallocation problem for this disk. At first, this CSP is created using the overloaded disk and all unclassified disks (i.e. those with a low level of load). However, if we fail to find a solution for this CSP, we expand it to include all disks classified as medium as well. In this manner, we process all disks until they all have satisfactory load levels (or until we discover we cannot find a solution to the problem).

Our hypothesis is that the larger (i.e. global) models suffer from scalability issues (the full problem more so than the restricted one). The localised model is not expected to suffer from such scalability issues, but does lead to lower quality solutions: we expect to use more moves to rectify the situation.

4.3 Implementation

We used the ThinkSmart Technologies Intellify platform [4] to build our system. Intellify is a Java platform that allows us to seamlessly integrate the OpenRules BRMS [5] as well as various CP solver implementations. The results we present here are obtained using the Constrainer [6] solver. We also have working versions using Choco [7] and an initial implementation of the forthcoming JSR-331 standard [8].

4.4 Results

The experimental data were generated using Java 1.6 on an Intel Core 2 Duo P8700, running at 2.53 GHz and with 4 GB of memory available (although none of the experiments required more than 512 MB). We allowed each experiment a maximum of 15 minutes of CPU time (i.e. 900 seconds).

Figures 2 and 3 show the results of our experiments, taking the average of the 10 instances for each number of disks tested. Notice that the figures represent

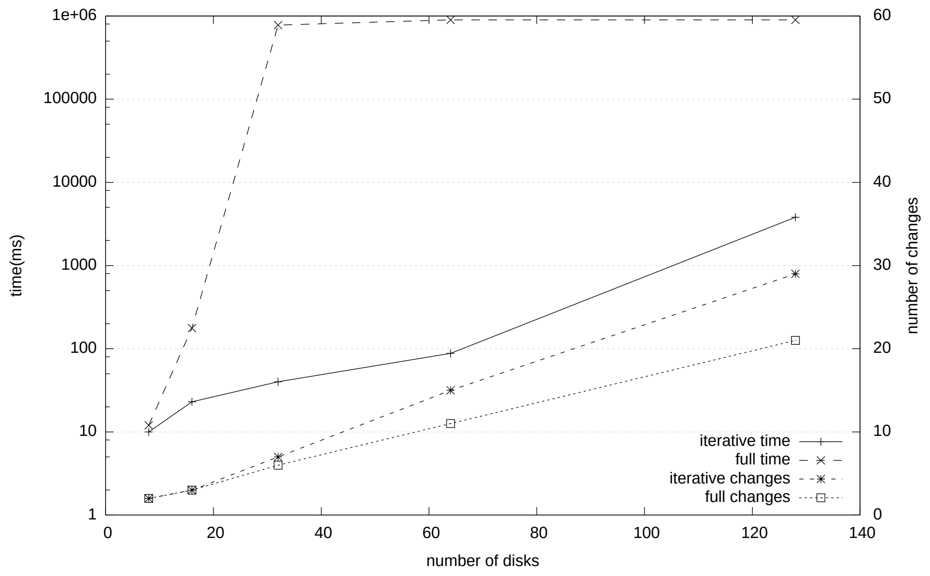


Fig. 2. Results with 12.5% of servers over loaded

both the CPU time required to solve the instance, as well as the quality of the resulting solution (as measured in the number of moves). The two figures show a similar pattern: when the number of disks increases, solving the whole problem as one quickly leads to performance issues, and we cannot prove that we have found the optimal solution within 15 minutes. On the other hand, when we deal with each overloaded disk in isolation (and finding an optimal result during each iteration), it takes less than 10 seconds for the largest of problems. Thus, this part of our hypothesis is confirmed by these experiments. A χ^2 analysis shows that this is a statistically significant difference, as shown in Table 4.

The other part of our hypothesis stated that we expect to find that by solving iteratively, we produce lower quality solutions, i.e. we require more changes to achieve the desired load levels. Again, the figures confirm this hypothesis, as does the χ^2 analysis (cf. Table 4).

A closer examination of the results, shows exactly why this happens. When we look at the overloaded disks in isolation, the only way to deal with the issue of one of the disks being overloaded is to move at least 1 application away from the disk, and replace it with another volume (either belonging to some other application with a lighter load, or even an unused volume from another disk).[5] Thus, 2 changes are required to deal with the issue. For example, consider 3 disks,

[5] Notice that the volumes in an array are of the same size. Disks are broken up into as many volumes of that size as possible (even if one or more of these will be unused) and thus no disks will have spare capacity to accommodate an additional volume. Hence, we always have to *swap* volumes.

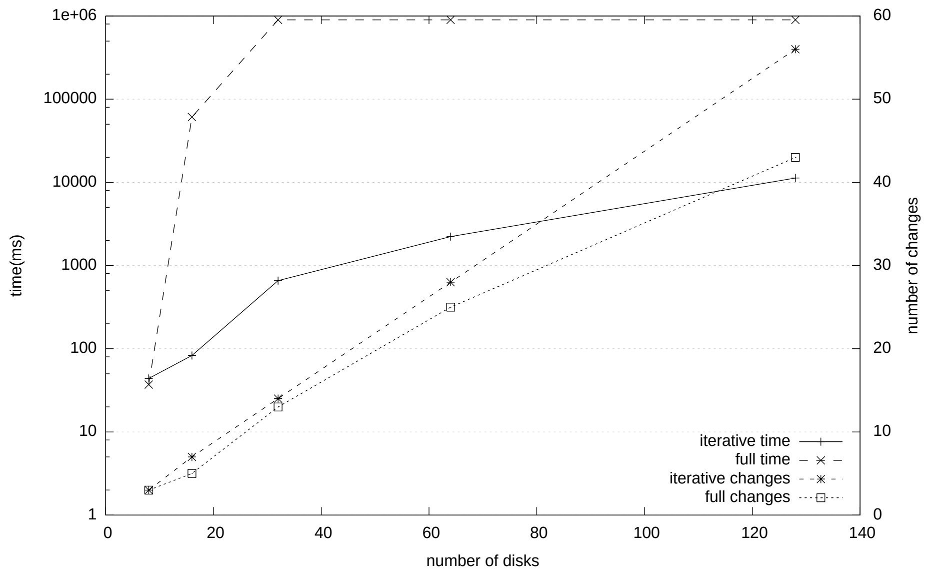


Fig. 3. Results with 25% of servers over loaded

d_1, d_2, and d_3, each with two applications on them with loads: $d_1 = \{40, 25\}$, $d_2 = \{40, 25\}$ and $d_3 = \{10, 10\}$. Suppose we want to achieve a maximum load of 60, and consider disk d_1 before disk d_2. The local problem generated to deal with d_1 consists of disks d_1 and d_3 (d_2 is not considered, since it's overloaded). It is straight-forward to see that we can achieve the objective by swapping the application with load 25 with one of the applications from disk d_3 (totaling 2 moves). The same holds when we consider d_2, and we achieve a total of 4 moves. Of course, in general, we may undo some of the moves we have made during an earlier iteration in order to satisfy the load on other disks, so we expect to see somewhere just under $2m$ changes given that there are m overloaded disks. (This number is confirmed by the graphs above.)

On the other hand, when we consider the solutions found when solving the whole problem at once, we observe that often, load is transferred between overloaded disks as well. In our previous example, for example, we can achieve a solution with only 3 moves: Move the application with load 40 from d_1 to d_3, move the one with load 25 from d_2 to d_1 and move the one with load 10 from d_3 to d_2. This requires that load is transferred between overloaded disks. Having a global view helps in detecting such situations, which explains the difference between the two approaches.

5 Related Work

There are a few instances of research in the combination of rules and constraints. Many of these works [9,10,11,12,13,14] relate to translating business rules into

Table 4. χ^2 analysis of the results, counting the number of problem instances one approach out-performs the other

	$^1/_{4th}$ overloaded		$^1/_{8th}$ overloaded	
	CPU time	Quality	CPU time	Quality
full model	4	41	7	33
iterative	46	0	43	0
χ^2	35.28	41	25.92	33
p	$\ll$0.01	$\ll$0.01	$\ll$0.01	$\ll$0.01

constraints and solving the problem with a single technology. This is not always applicable since business rules on their own can be used to make decisions using a different technology based on the RETE algorithm [15]. Perhaps closest to the research presented here is the work by Bousonville et al. [16], who use Business Rules to generate the data input to a predefined optimisation model. In particular, they say, "we do not provide direct access to decision variables so that only predefined constraints are possible". This is quite in contrast to what is demonstrated here. We indeed dynamically create constrained variables and post additional constraints from within the business rules environment. Rules are integral to solving the problem, not just as a pre-processor, but as an active component.

LAURE [17] (and its successor CLAIRE [18]) is a system which allows the use of rules to guide the optimisation search. *Constraint Handling Rules* (CHRs) are a more general approach to solving CSPs using rules. An overview of CHR approaches can be found in [19].

The automatic generation of dispatch rules (similar to business rules) for a packaging problem [20] through a CP optimisation model shows another hybrid configuration. However, there is a loose coupling between the technologies. Here an optimisation model, reflecting the global demands over a time period, is used to choose the parameters for a set of rules describing which task and which packaging machine to consider next.

6 Conclusions and Future Work

In this paper, we describe a hybrid system that uses both a Business Rules Management System and a Constraint Programming solver to solve resource reallocation problems in large data centres. Such problems arise because the applications assigned to a server fluctuate in their requirements for, e.g. CPU power, memory, and bandwidth. When the demands on a server are larger than what it can provide, the performance degrades, and a solution has to be found through moving some of the workload to other servers.

In our solution, we use a BRMS to analyse the states of the servers over time. Due to stability reasons, we only want to move applications when a pattern has been established over several successive measurements, and the BRMS is used to describe under which exact conditions servers are considered to be overloaded.

Once the BRMS has established that one or more servers are overloaded, the rules that have fired are also used to construct a CSP corresponding to the problem. This is a distinguishing feature of our approach, as heretofore rules have only been used to setup parametrised models or to provide input to a CSP model. In contrast, our system allows the rules to create and access directly the variables and constraints in the CSP. This gives the rule administrator full control over how the problem is generated and solved, hence allowing the solution to be customised to the specific circumstances in their data centre.

The combination of Business Rules and CP is a powerful approach to the problem. For Business Rules, finding an optimal solution is problematic and could conceivably lead to many rules which are difficult to maintain. For Constraint Programming, the creation of the instance of the problem model would require much programming, and small changes to the underlying logic would require reprogramming that part of the solution. By passing each challenge to a different technology and linking them together at a low level delivers a fast, easily maintainable solution.

We have tested the system using data from a large data storage centre. The results show that while achieving optimal solutions is impractical for large problems (due to the time involved in constructing those optimal solutions), we are able to solve the problem within seconds by iteratively dealing with each overloaded resource in turn. However, this comes at a price, as the quality of the solution found is lower.

There are several areas for improvement to the current system. First of all, we want to examine the trade-off between solving iteratively and solving the problem as a whole. We identified specific conditions that improve the solution quality when solving the full problem (i.e. moving applications between overloaded disks in addition to the moving of applications between overloaded disks and those that are not overloaded that happens in the iterative solution strategy). By solving slightly larger problems, we may be able to achieve the best of both. This could even be set by a rule in the BRMS, allowing the data centre administrators to make the trade-off between solution quality and speed of achieving the solution.

Secondly, energy consumption is becoming a major issue for data centres. For this reason, we want to introduce the ability to recommend shutting down servers when not all servers are required, and *vice versa*, to recommend turning some additional servers on when there is a serious resource deficiency. This can be modeled already within the formal model of Section 2 (by introducing all-consuming dummy processes that are bound to servers when they are not available), but we have not explored this aspect using real data yet.

References

1. Barroso, L., Hölzle, U.: The datacenter as a computer: An introduction to the design of warehouse-scale machines. Synthesis Lectures on Computer Architecture **4** (2009) 1–108
2. `http://www.vkernel.com/`

3. Herbst, H., Knolmayer, G., Myrach, T., Schlesinger, M.: The specification of business rules: A comparison of selected methodologies. In: Tools for the Information System Life Cycle. (1994)
4. `http://www.thinksmarttechnologies.com/`
5. `http://www.openrules.com/`
6. `http://www.constrainer.sourceforge.net/`
7. `http://www.emn.fr/z-info/choco-solver/`
8. `http://4c110.ucc.ie/cpstandards/index.php/en/standards/java/jsr-331`
9. Kameshwaran, S., Narahari, Y., Rosa, C., Kulkarni, D., Tew, J.: Multiattribute electronic procurement using goal programming. European Journal of Operational Research **179** (2007) 518 – 536
10. Carlsson, M., Beldiceanu, N., Martin, J.: A geometric constraint over k-dimensional objects and shapes subject to business rules. In: Proceedings of the 14th International Conference on Principles and Practice of Constraint Programming (CP-2008). (2008) 220 – 234
11. Fages, F., Martin, J.: From rules to constraint programs with the Rules2CP modelling language. In: Recent Advances in Constraints. (2009)
12. Feldman, J., Korolov, A., Meshcheryakov, S., Shor, S.: Hybrid use of rule and constraint engines (patent no: WO/2003/001322). (World Intellectual Property Organisation)
13. Feldman, J., Freuder, E.: Integrating business rules and constraint programming technologies for EDM. In: The 11th International Business Rules Forum and The First EDM Summit. (2008)
14. O'Sullivan, B., Feldman, J.: Using hard and soft rules to define and solve optimization problems. In: The 12th International Business Rules Forum. (2009)
15. Forgy, C.: Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence **19** (1982) 17–37
16. Bousonville, T., Focacci, F., Pape, C.L., Nuijten, W., Paulin, F., Puget, J.F., Robert, A., Sadeghin, A.: Integration of rules and optimization in plant powerops. In: Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming. (2005) 1–15
17. Caseau, Y., Koppstein, P.: A cooperative-architecture expert system for solving large time/travel assignment problems. In: Proceedings of the International Conference on Database and Expert Systems Applications. (1992) 197–202
18. Caseau, Y., Laburthe, F.: CLAIRE: Combining objects and rules for problem solving. In: Proceedings of the JICSLP'96 Workshop on multi-paradigm logic programming. (1996)
19. Sneyers, J., van Weert, P., Schrijvers, T., de Koninck, L.: As time goes by: Constraint handling rules, a survey of chr research from 1998 to 2007. Theory and practice of logic programming **10** (2010) 1–48
20. van der Krogt, R., Little, J.: Optimising machine selection rules for sequence dependent setups with an application to cartoning. In: Proceedings of the 13th IFAC Symposium on Information Control Problems in Manufacturing. (2009) 1148–1153