Proceedings of the *9th IEEE/IFIP Network Operations and Management Symposium (NOMS 2004)*, Seoul, Korea, April 2004

# The CHAMPS System: Change Management with Planning and Scheduling

*A. Keller, J.L. Hellerstein, J.L. Wolf, K.-L. Wu, V. Krishnan*
*IBM T.J. Watson Research Center*
*P.O. Box 704, Yorktown Heights, NY 10598, USA*
*{alexk|hellers|jlwolf|klwu|vijaya33}@us.ibm.com*

**Abstract**

Change Management is a process by which IT systems are modified to accommodate considerations such as software fixes, hardware upgrades and performance enhancements. This paper discusses the CHAMPS system, a prototype under development at IBM Research for **CHA**nge **M**anagement with **P**lanning and **S**cheduling. The CHAMPS system is able to achieve a very high degree of parallelism for a set of tasks by exploiting detailed factual knowledge about the structure of a distributed system from dependency information at runtime. In contrast, today's systems expect an administrator to provide such insights, which is often not the case. Furthermore, the optimization techniques we employ allow the CHAMPS system to come up with a very hiqh quality solution for a mathematically intractable problem in a time which scales nicely with the problem size. We have implemented the CHAMPS system and have applied it in a TPC-W environment that implements an on-line book store application.

**Keywords**

Change Management, Software Deployment, Workflows, Plan, Schedule Optimization

## 1. Introduction

The goal of Change Management is "to ensure that standardized methods and procedures are used for efficient and prompt handling of all changes, in order to minimize the impact of Change-related Incidents upon service quality, and consequently to improve the day-to-day operations of the organization" [10]. The change management process starts with the submission of a **Request For Change (RFC)**, which is viewed as a **job** in scheduling terms. Many RFCs may be submitted concurrently. The RFC describes what is to be done, usually in terms of hardware/software artifacts to change (deploy, install, configure, uninstall), as well as the deadline by which the change needs to be completed. Examples include changing the schema of a database table in a running application and installing a new release of a web application server in a multi-tiered eCommerce system. An important observation is that many changes are not explicitly included in the RFC. Rather, they are merely implied. For example, applications must be recompiled if they use a database table whose schema is to change. Such implicit changes are a result of various kinds of relationships, such as service dependencies and resource sharing.

The discussion of dependencies is central to our work and so deserves some elaboration. Figure 1 displays dependencies in an on-line book store application used in the

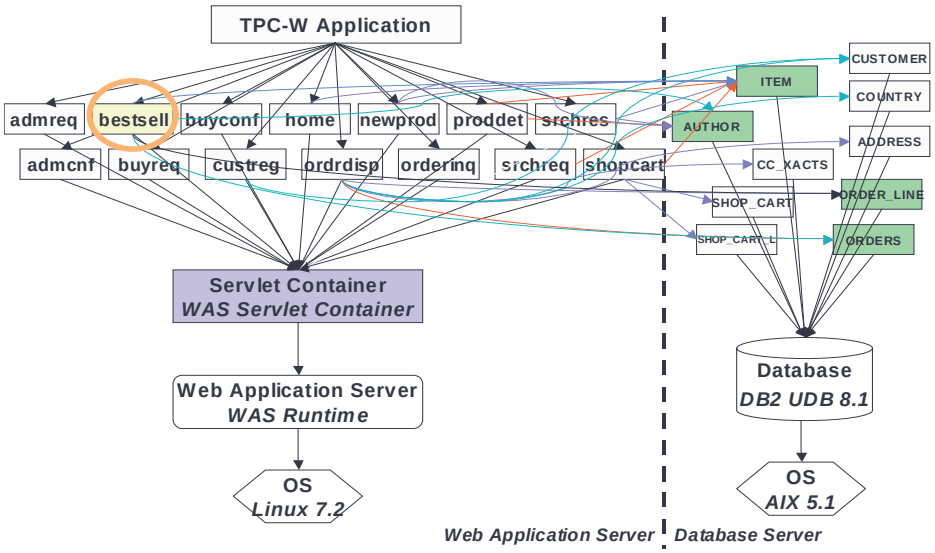


**Figure 1:** Dependency Structure of the TPC-W benchmark implementation

Transaction Processing Council's Web Commerce (TPC-W) benchmark [17]. This can be viewed as a two tiered application consisting of a web application server and a database server. The TPC-W application consists of fourteen servlets and ten database tables. The servlets are hosted by a Servlet Container, which depends on the runtime environment of a Web Application Server. This, in turn, uses the operating system services. A similar hierarchy exists in the database server as well: The ten database tables depend on the services of the database management system, which in turn uses operating system services. Thus, a request to install the BestSellers Service operates bottom-up on both the application and database servers. For example, on the web application server, we proceed by installing the operating system, the server runtime environment, the servlet container, and then the `bestsell` servlet (highlighted by the circle in the Figure) that implements the BestSellers service. If one or more of these steps has already been completed, then we immediately proceed with the next step.

There is a well established methodology for change management (e.g., [10, 5]), which consists of the following steps:

1. Assess the impact of changes in terms of the resources and services affected.
2. Create a Change Plan that dictates how the change should be implemented.
3. Verify the Change Plan (e.g., through review by a change management team in discussions with affected departments).
4. Test the Change Plan by doing "dry runs" especially for very disruptive and/or high risk changes.
5. Ultimately implement the change by executing the plan.

In practice, steps (1) and (2) turn out to be particularly difficult. The **Change Plan** consists of a set of **Tasks** needed to complete the RFC such as "bring down application server 1"

and “copy x.ini to server 2”. The plan itself specifies the partial order of tasks. Note that items (2)-(5) relate to creating, evaluating, and implementing the Change Plan.

It turns out that impact analysis (step (1) above) also requires a Change Plan. To see this, consider Figure 1 and suppose that: (a) there is an RFC that modifies the schema of the ITEM Table; and (b) the installation places a high priority on the BestSellers Service. Assessing the potential impact of the RFC means knowing when BestSellers, implemented by the `bestsell` servlet, will be unavailable. To do this, we must know the Change Plan and the timing of tasks in the plan (e.g., when must `bestsell` be taken off-line and when will the service be restored). The former is the outcome of planning and the latter is produced by scheduling.

The importance of change management is underscored by a recent study showing that operator errors account for the largest fraction of failures of Internet services [14] and hence properly managing changes is critical to availability. Many different tools exist for handling the mechanics of software and patch distribution for end user systems and servers, such as Microsoft Windows Update [12] and Tivoli’s Configuration Manager [8], and more broadly systems for agent-based configuration management such as cfengine [4, 16]. Too often, these solutions do not take a “holistic” approach [18] in that they focus on desktop systems rather than broadly on a set of interconnected servers providing end-to-end services. A holistic approach is facilitated by constructing a Change Plan, and workflow descriptions are a good representation for such a plan. Indeed, there has been interest in using workflow technologies to coordinate large scale efforts such as change management [11], and in the application of project management techniques as well [9]. However, we are unaware of any system that dynamically constructs a Change Plan that takes into account the impact of changes on service levels, although some have advocated the value of doing so [13]. We note in passing that constructing a Change Plan requires knowledge of service dependencies. This means discovering dependencies, as in [2], as well as representing dependencies, as in [6] and [19].

This paper is about automating the construction of Change Plans as a first step towards broader automation of change management. The design of the CHAMPS system follows a ”best of breed” approach by leveraging existing tools and techniques (such as workflows, project management, and mathematical scheduling theory) for an integrated Change Management solution. It consists of a **Task Graph Builder (TGB)** and a **Planner and Scheduler (P&S)**. The TGB determines the temporal and location constraints of tasks needed to complete the RFC. Such a **Task Graph** is constructed based on dependency information and installation policies (e.g., time-of-day considerations for making changes). The P&S uses the task graph to construct the **Change Plan**. Planning specifies the partial order of tasks and binds logical to physical resources (e.g., selecting which of several available machines should become the application server). Scheduling determines the times at which actions take place.

The remainder of the paper is structured according to several features that distinguish the CHAMPS system from the current state of the art:

An important property of the CHAMPS system is that it enables an administrator to extend and modify the data within both Task Graphs and Change Plans with common off-the-shelf tools. Using a general-purpose workflow language, such as BPEL4WS [1] for

expressing both Task Graphs and Change Plans is key for achieving this. Our description of the CHAMPS architecture in section 2 provides more details.

Second, by splitting the overall process into two parts (Task Graph Building, Planning & Scheduling) and thus binding workflows to target systems at a fairly late stage in the process, we achieve a high degree of reusability for the information dealing with the software artifacts subject to a change. This is due to the fact that Task Graphs are not tied to a specific environment and are thus neither influenced by the characteristics of the target systems, nor financial constraints (expressed, e.g., in SLAs). Section 3 describes the Task Graph Builder.

Third, our approach formulates the planner and scheduler as an optimization problem, which allows us to apply mathematical scheduling theory. The goal is to maximize the profits associated with performing the jobs associated with a selected subset of RFCs. The profit for each RFC can be expressed as the value of performing the job minus the associated costs. The generic nature of our objective function incorporates a large number of practical variants as special cases. In addition, the resulting P&S is required to obey a variety of realistic temporal, location-specific and other types of constraints. This formulation is mathematically intractable in the sense that it is effectively impossible for our (or any other) algorithm to find an exact optimal solution in a reasonable amount of time. The optimization techniques we employ allow us to come up with a very hiqh quality solution in a time which scales nicely with the problem size. Section 4 discusses the Planner & Scheduler. Our conclusions and future directions are contained in section 5.

## 2. Architecture of the CHAMPS System

Figure 2 depicts the architecture of the CHAMPS system as well as the interactions between the different components. We begin our description with the submission of a new RFC by the administrator (depicted in the upper left part of the Figure). An RFC contains the name of the software artifact(s) that need to be changed, the name(s) of the target system(s) and the requested operation (e.g., "update the `orderDisplay` and `buyConfirmation` servlets, as well as the `CC_XACTS` database table"). In addition, the RFC contains the deadline (time/date) by which the change must be completed, as well as its maximum allowable duration (e.g., a maintenance interval with a length of 2 hours, ending at 5am). Note that an RFC is *declarative* by stating *what* needs to be accomplished, but leaves the *procedural* details (*how* the change is carried out) open.

Based on the submitted RFC, the **Task Graph Builder** determines the allowable order of the tasks that are necessary to fulfill the RFC. To do so, it exploits two sources of dependency information:

1. The first source are **Deployment Descriptors** that annotate software packages, which reside in the Package Repository (lower left part of the Figure). Deployment descriptors (such as the ones used by Linux RPM or AIX installp packages) provide meta-information, gathered at build time (and preferably automatically generated by the development tools), about a software package, such as identifying and version data, and dependency information. This dependency information lists the pre-requisites (packages that must be present on the system for an installation to succeed), the co-requisites

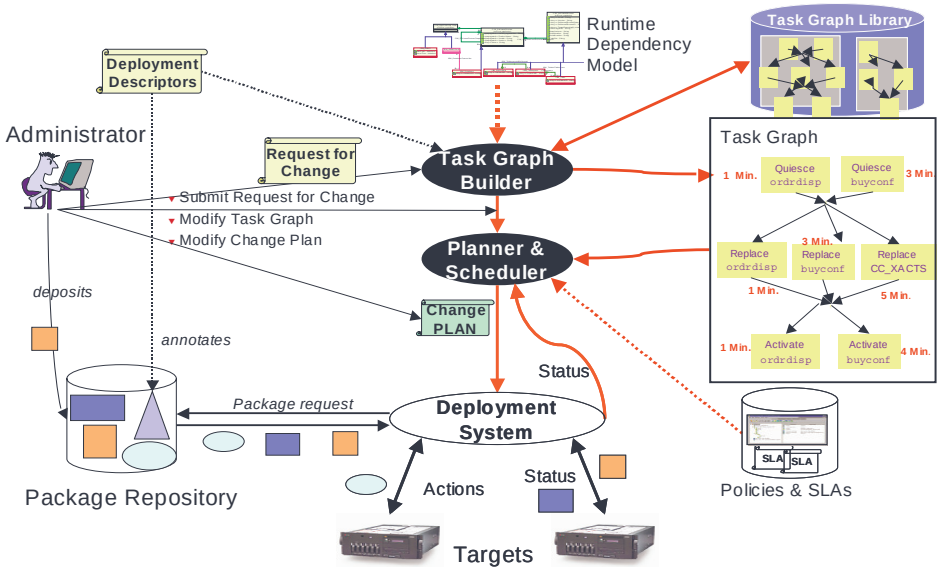


**Figure 2:** Architecture of the CHAMPS System

(packages that must be jointly installed) as well as ex-requisites (packages that must be removed prior to installing a new package). The details on the deployment descriptors we use, and the mechanism for collecting and consolidating this information across multiple systems have been described in [6].

2. In contrast to the dependency information captured in deployment descriptors, a **Runtime Dependency Model** captures dependencies that typically cross system boundaries. In [2], we have demonstrated that it is possible to discover runtime dependencies between the servlets hosted on a web application server and the back-end database tables of a TPC-W testbed with a very high degree of confidence (41 out of 42 possible dependencies were identified) by applying a perturbation-based approach.

With this information, the Task Graph Builder is able to determine the steps of a change as well as the order in which they have to be carried out. We call a representation of such information **Task Graph**. A task graph is an abstract workflow, consisting of tasks and precedence constraints that link these tasks together. In addition, the task graph comprises time estimates for every task, which may have been obtained from previous deployments. Note that the information stored within a task graph is specific to a given combination of software artifacts, but completely decoupled from the target systems and their characteristics (e.g., CPU speed, RAM, total/available diskspace). Consequently, Task Graphs can be stored in a Task Graph Library (depicted in the upper right part of Figure 2) and subsequently reused, modified and aggregated by an administrator using common off-the-shelf workflow editors.

The Task Graph is then consumed by the **Planner & Scheduler (P&S)**. Its purpose is to assign tasks to the systems specified in the RFC according to additional monetary

and technical constraints, such as Service Level Agreements (SLAs) and Policies. Further, it computes (according to various administrator-defined criteria, detailed below) a **Change Plan** that includes deadlines and maximizes the degree of parallelism for tasks according to precedence and location constraints expressed in the Task Graphs. Again, we use a workflow language (BPEL4WS, see section 3.2) to express the Change Plan, which facilitates further manual modifications and extensions by an administrator, if needed.

Once the Change Plan has been computed by the P&S, it is input to the **Deployment System**, which retrieves the required software packages from a Package Repository, and rolls out the requested changes to the Targets in the order specified in the plan. An important part of the deployment process is the capability of the deployment system to keep track of how well the roll-out of changes progresses on the targets, and to feed this status information back into the P&S. Being aware of the current deployment status enables the P&S to track the actual progress against the plan and perform on-line plan adjustment (by re-computing the change plan) in case the deployment runs behind schedule. In addition, such a feedback mechanism can be used to gain an understanding on how long it takes to complete a task. This knowledge can then be stored in the Task Graphs to gain realistic task time estimates for future deployments.

## 3. Task Graph Builder

The purpose of the Task Graph Builder (TGB) is to create reusable abstract workflows (Task Graphs) from existing dependency descriptions. Task graphs describe the partial order in which tasks need to be carried out to transition a system from a workable state into another workable state. In order to achieve this, a task graph contains information about:

- The type of change to be carried out, e.g., install, update, configure, uninstall,
- the roles and names of the software artifacts that are subject to a change (either directly specified in the RFC, or determined by the TGB),
- the temporal and location constraints that may exist between tasks, based on artifact dependency information,
- an estimate of how long every task is likely to take, based on the results of previous deployments. This is needed to estimate the impact of a change in terms of downtime.

### 3.1 Building Task Graphs from Software Artifact Dependencies

In previous work [6], we have described a system for identifying and tracking dependencies between the software artifacts of distributed systems, with a focus on fault management and problem determination. Its core component is a Dependency Query Facility that provides a user with an API to execute operations for (recursively) traversing a dependency graph from the top to the bottom (drill–down), or in the opposite direction (drill–up).

The Task Graph Builder exploits this previous implementation by invoking the Dependency Query Facility and evaluating the returned dependency graph to determine whether tasks implied by a change must be carried out sequentially, or whether some of them can be parallelized. The existence of a dependency between two artifacts indicates that a temporal/location constraint must be addressed. If a temporal constraint exists between two

tasks, they need to be carried out within a **sequence**. Any task may have zero or more incoming and outgoing links. If two tasks share the same predecessor and no temporal constraints exist between them, they can be executed concurrently within a **flow**. The outermost container for grouping tasks and their constraints on a per-host basis is a sequence. Grouping on a per-host basis is important because the actual deployment could happen either push-based (triggered by the deployment system) or pull-based (deployment is initiated by the target systems)

We have found that different types of changes require different traversals through the dependency models: A request for a new installation of a software artifact leads the TGB to invoke a recursive drill-down operation on the Dependency Query Facility to determine which artifacts must already be present before a new artifact can be installed. On the other hand, a request for an update, or an uninstall of an artifact leads to the invocation of a recursive drill-up query to determine the artifacts that will be impacted by the change.

In order to be useful for change management, we need to refine the notion of a temporal constraint by extending the workflow management formalisms. More specifically, we need to add three additional types of temporal constraints to the one temporal constraint type that is available in typical workflow management systems. These four constraint types are widely used in GANTT charts within the Project Management discipline [9], but are applicable to Change Management as well:

- Finish-to-Start (FS): This temporal constraint expresses that task A must finish before task B can begin and is the default constraint in workflow management systems. An example in the TPC-W context is that a servlet container must be running (i.e., the task of starting it must be completed) before a new servlet can be deployed to it.
- Start-to-Start (SS): Task B cannot start until task A does. An example of this constraint type are nested transactions and units of work.
- Finish-to-Finish (FF): Task B cannot finish before task A does. Example: One cannot shut a system down if the web application server is still running.
- Start-to-Finish (SF): Task B cannot finish until task A starts. Example: a failover server cannot be taken offline before the main server is up again. Note that there is a subtle difference between this constraint type and the aforementioned FS constraint type, because here the start of a task determines the end of its predecessor (in the simpler FS case, the start of a task depends on the ending of its predecessor).

As an example, Figure 3 depicts a (simplified) task graph for installing the `bestsell` servlet of the TPC-W Internet storefront, described in section 1. Based on the dependencies between the artifacts and assuming that an installation request has been submitted, we traverse the dependency model by means of a drill-down, which yields the order of the tasks, along with their temporal constraint types: in this example, they happen to be FS-type constraints, however, any of the three other constraint types can be accomodated as the constraint type is an attribute of a link and thus embedded in the Task Graph.

In this example, we can observe three interesting results: First, the fact that the constraints crossing the system boundaries apply to tasks that happen at a fairly late stage in the install process allows the deployment system to achieve a high degree of parallelism because the installation happens on different systems. Second, the evaluation of the dependencies and their constraints yields that the most time-consuming tasks (installation

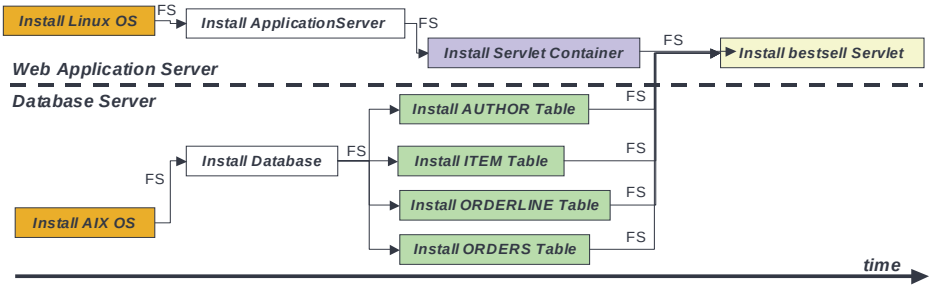


**Figure 3:** Task Graph for installing the TPC-W `bestsell` Servlet

of application server and database server) can be carried out completely in parallel. Third, another opportunity for concurrent installation is the fact that the four database tables share the same antecedent and do not have dependencies among each other. Thus, they can be concurrently installed as well. However, it should be noted that local parallelism is difficult to exploit on current operating systems as few multithreaded installers exist as of today. In addition, some operating systems require exclusive access to shared libraries during installation.

### 3.2 Expressing Task Graphs in the BPEL4WS Workflow Language

We will now describe how the task graph described in the previous section can be represented in a format suitable for interpretation by workflow editors and engines that may be embedded in the deployment system. Further, we assume push-based deployment. After having evaluated several alternatives, we decided to use the *Business Process Execution Language for Web Services (BPEL4WS)* [1], a recent standardization proposal for a workflow language based on XML Schema, with a focus on Web Services. BPEL4WS provides a very flexible mechanism for composing workflows and business processes and provides all the necessary constructs we have identified in the previous section. In addition, several tools for editing BPEL4WS workflows and checking their syntax, as well as a workflow engine, are available to the general public (they can be downloaded from http://www.alphaworks.ibm.com/tech/bpws4j). An administrator would therefore be able to use these off-the-shelf tools to manually modify any task graph the TGB generates, according to his needs.

We would like to stress the point that the structure of the BPEL4WS representation is essentially identical to Figure 3, which gives a hint on how straightforward the mapping of *local* dependencies (i.e., the ones confined to a single system) into a BPEL4WS workflow is. Without hardly any further information, a workflow engine would be able to execute the tasks within the two sequences in the proper order. However, the both outermost sequences would be completely decoupled as well, as they are contained within a flow. This, however, would be incorrect, as it would ignore the *cross-system* temporal constraints, namely between the four database tables and the `bestsell` servlet. Without this information, a deployment system would allow an administrator to start the `bestsell` servlet without having checked if the four underlying database tables are actually in-

stalled. This problem can be fixed in the following way: One needs to insert a temporal constraint from each of the four database tables to the `bestsell` servlet. We demonstrate this for the `Install_ITEM_Table` task, whose syntax is listed below:

```
1 <invoke name="Install_ITEM_Table" joinCondition=""
2            partner="DB2 UDB v8.1" portType="tns:Install_ITEM_TablePT"
3            operation="Install_ITEM_Table" inputVariable="Install_ITEM_Table">
4    <source linkName="Install_ITEM_Table_to_Install_BestSellers_Servlet"/>
5    <target linkName="Install_DB2_UDB_v8.1_to_Install_ITEM_Table"/>
6 </invoke>
```

Lines 1-3 of the listing declare the task itself, along with references to the interface definition and other supporting information. In line 4, we specify the temporal constraint between the `Install_ITEM_Table` and the `Install_BestSellers_Servlet` tasks by means of the BPEL4WS `source` construct. Note that BPEL4WS requires a link to be specified in both tasks it connects. Therefore, one needs to specify this link as well within the `Install_BestSellers_Servlet` task by means of the `target` construct, which indicates that this task has an ”incoming” link. For demonstration purposes, we have inserted a redundant (because `Install_ITEM_Table` will be carried out after `Install_DB2_UDB_v8.1` anyway, since it is part of a sequence) link `Install_DB2_UDB_v8.1_to_Install_ITEM_Table` in line 5 of the listing above. By inserting an additional link, we make sure that the deployment system proceeds with executing the task `Install_BestSellers_Servlet` only *after* the task `Install_ITEM_Table` has completed.

It should be noted that BPEL4WS provides an exhaustive set of functions to declare workflows out of which we have only shown a very minor subset for illustrative purposes. Examples for further capabilities are constructs for specifying the execution semantics of links, or the introduction of timer events that allow tasks to be carried out based on elapsed time intervals. Especially the latter is important for Change Plans, whose construction by the P & S is discussed in the following section.

## 4. Planner and Scheduler

### 4.1 Overview

In this section we provide an overview of the CHAMPS Planner and Scheduler (P&S). Given the space constraints we choose to focus on the optimization problem formulation, limiting our discussions for now of the solution techniques. We will instead emphasize the generic nature of our formulation, particularly as it pertains to the objective function. Our formulation allows a large collection of interesting and useful variants to be solved as special cases. Naturally, this generic formulation causes the overall optimization problem to be that much harder, but most reasonable special cases fall into the NP-hard category anyway [7]. The P&S therefore employs heuristic solution techniques, achieving high quality but not typically strictly optimal solutions. Then we shall provide a specific example of P&S input and output.

The CHAMPS P&S is hierarchical in nature. Each RFC corresponds to a *job* which may or may not be done during the particular window of time, called the *change window*, under consideration. Associated with each job are a set of *tasks* which are interrelated by temporal and location-specific dependencies arising from the TGB, and a variety of other constraints as well. The P&S solution respects this hierarchy, effectively decoupling the

problems of scheduling the various tasks which comprise each job from the problem of scheduling the jobs themselves.

Roughly speaking, the CHAMPS P&S attempts to maximize the value associated with the jobs that will be done within a given change window minus the total costs of jobs that will be done (and thus optimizing the overall profits), while satisfying the following extensive set of constraints:

1. Precedence constraints among tasks within a job are respected. In other words, if task $i_1$ of job $j$ is required to finish before task $i_2$ of job $j$ starts, the scheduler will enforce this. (In section 3.1, we have called this a *finish-to-start (FS)* constraint.)
2. Similarly, *start-to-start (SS)* constraints are respected for each job. Thus task $i_1$ of job $j$ may be required to start before task $i_2$ of job $j$ starts.
3. *Start-to-finish (SF)* constraints are respected for each job. Thus task $i_1$ of job $j$ may be required to start before task $i_2$ of job $j$ finishes.
4. *Finish-to-finish (FF)* constraints are respected for each job. Thus task $i_1$ of job $j$ may be required to finish before task $i_2$ of job $j$ finishes.
5. Tasks only get assigned to *acceptable* servers. The list of acceptable servers may be as big or as small as desired. In particular, the list may consist of all the servers or just one. It might also consist of a specific class of servers.
6. *Colocation (CL)* task/server assignment constraints are met for all jobs. Thus, tasks $i_1$ and $i_2$ of job $j$ may be required to be performed on the same server.
7. *Exlocation (EL)* task/server assignment constraints are met for all jobs. Thus, tasks $i_1$ and $i_2$ of job $j$ may be required to be performed on different servers.
8. Resource *capacity* constraints are met on each server. These constraints might be used to meet CPU utilization, memory and disk capacities, for example.
9. Jobs get done if they are *required*. For example, those jobs with a deadline that falls within the change window must be performed. Others might be postponed for the time being, and thus might be regarded as optional. They might, for example, get done during a subsequent change window.
10. Each task of a job that gets done is assigned to a single server.
11. No server can work on more than one task at any time.
12. All tasks on all jobs that get done must be performed during the change window.

In the above list, constraints (1)-(4) are temporal, and arise from the TGB. Constraints (5)-(8) are location-specific, and may arise from the RFCs themselves, by virtue of policy, or from the system state. Constraint (9) ensures that required jobs get scheduled. This might include jobs whose deadlines fall within the change window, as specified by the RFC. Constraints (10)-(12) are technical but standard scheduling requirements.

### 4.2 Optimization Problem Formulation

In order to formalize these concepts we will need some notation. So let $\{1, ..., J\}$ be the set of jobs, indexed by $j$. Let $\mathcal{I}_j = \{1, ..., I_j\}$ be the set of tasks in job $j$, indexed by $i$. Let $\prec_j$ denote the precedence relation for job $j$ derived from the TGB. (Without loss of generality assume tasks are in topological order.) Let $\{1, ..., P\}$ be the set of servers, indexed by $p$, and $\{1, ..., R\}$ be the set of resource types, indexed by $r$. Let $T$ denote the length of the change window.

There will be a set $\mathcal{K}_j = \{1, ..., K_j\}$ of cost summands for job $j$, indexed by $k$. Each summand $k$ will last from the start of task $\alpha_{j,k} \in \mathcal{I}_j$ to the end of task $\beta_{j,k} \in \mathcal{I}_j$. We will integrate these cost summands for this duration of time, and then add these integrals together to obtain the cost component of the objective function. Special cases include costs which run through the duration of the entire job ($\mathcal{K}_j = 1$, $\alpha_{j,1} = 1$, $\beta_{j,1} = I_j$) and costs which are task-specific ($\mathcal{K}_j = \mathcal{I}_j$. $\alpha_{j,k} \equiv \beta_{j,k} \equiv k$). Define $X_j$ to be 1 if job $j$ must be done, and 0 otherwise. The precedence relation $\prec_j$ for job $j$ yields a finish/start set $FS_j \subset \mathcal{I}_j \times \mathcal{I}_j$: $(i_1, i_2) \in FS_j \Leftrightarrow i_1 \prec_j i_2$. Similarly there is a start/start set $SS_j \subset \mathcal{I}_j \times \mathcal{I}_j$, a start/finish set $SF_j \subset \mathcal{I}_j \times \mathcal{I}_j$ and a finish/finish set $FF_j \subset \mathcal{I}_j \times \mathcal{I}_j$. (Of these four types of temporal constraints, the precedence, or finish-to-start, constraints are most common.) Let $A_{i,j}$ denote the acceptable server set for task $(i, j)$. Let $CL_j \subset \mathcal{I}_j \times \mathcal{I}_j$ be the colocation set for job $j$, and $EL_j \subset \mathcal{I}_j \times \mathcal{I}_j$ be the exlocation set. Let $t_{i,j,p}$ denote the execution time of task $(i, j)$ if assigned to server $p$. (That is, we assume the servers are heterogeneous.) Let $W_{p,r}$ denote the available capacity on server $p$ of resource $r$. Assume that the net effect on resource $r$ utilization of doing task $(i, j)$ on server $p$ is $w_{i,j,r}$. Let the value of doing job $j$ be $V_j$. On the other hand suppose that the $k$th cost summand for doing job $j$ at time $t$ is given by $C_{j,k}(t)$. (We typically assume that this function is constant within given intervals, though this assumption is not strictly necessary.)

The P&S employs three types of decision variables: The first two are binary, namely:

$$x_j = \begin{cases} 1 & \text{if job } j \text{ is done} \\ 0 & \text{otherwise} \end{cases}$$

and

$$a_{i,j,p} = \begin{cases} 1 & \text{if task } (i, j) \text{ is done on server } p \\ 0 & \text{otherwise} \end{cases}$$

The last is a real variable: $s_{i,j}$ is the start time of task $(i, j)$. Together these describe *whether* or not the job will be done, and, if done, *where* and *when* its various tasks will be performed.

The following dependent variables can be derived easily from these, and make the overall optimization problem easier to formulate: The completion time of task $(i, j)$ if done on server $p$ is given by $f_{i,j,p} = s_{i,j} + t_{i,j,p}$. And there is an execution indicator function for task $(i, j)$ given by

$$Z_{i,j,p}(t) = \begin{cases} 1 & \text{if } a_{i,j,p} = 1, s_{i,j} \le t \le f_{i,j,p} \\ 0 & \text{otherwise} \end{cases}$$

At last we can formulate our optimization problem, as follows: Maximize

$$\sum_j V_j x_j - \sum_j \sum_k \sum_i \sum_p a_{\beta_{j,k},j,p} \int_{s_{\alpha_{j,k},j}}^{f_{\beta_{j,k},j,p}} C_{j,k}(t) dt$$

such that

$$f_{i_1,j,p} \le s_{i_2,j} \text{ if } (i_1, i_2) \in FS_j, x_j = 1, a_{i_1,j,p} = 1 \; \forall j \tag{1}$$

$$s_{i_1,j} \le s_{i_2,j} \text{ if } (i_1, i_2) \in SS_j, x_j = 1 \; \forall j \tag{2}$$

$$s_{i_1,j} \le f_{i_2,j,p} \text{ if } (i_1, i_2) \in SF_j, x_j = 1, a_{i_2,j,p} = 1 \; \forall j \tag{3}$$

$$f_{i_1,j,p_1} \leq f_{i_2,j,p_2} \text{ if } (i_1,i_2) \in FS_j, x_j = 1, a_{i_1,j,p_1} = 1, \quad (4)$$

$$a_{i,j,p} = 0 \text{ if } p \notin A_{i,j} \; \forall (i,j) \quad (5)$$

$$a_{i_2,j,p} = 1 \text{ if } a_{i_1,j,p} = 1, x_j = 1, (i_1,i_2) \in CL_j \; \forall j \quad (6)$$

$$a_{i_2,j,p} = 0 \text{ if } a_{i_1,j,p} = 1, x_j = 1, (i_1,i_2) \in EL_j \; \forall j \quad (7)$$

$$\sum_i \sum_j a_{i,j,p} w_{i,j,r} \leq W_{p,r} \; \forall p, r \quad (8)$$

$$x_j \geq X_j \; \forall j \quad (9)$$

$$\sum_p a_{i,j,p} = x_j \; \forall i \quad (10)$$

$$\sum_i \sum_j Z_{i,j,p}(t) \leq 1 \; \forall p, t \quad (11)$$

$$0 \leq s_{i,j} < f_{i,j,p} \leq T \text{ if } a_{i,j,p} = 1 \; \forall i, j. \quad (12)$$

The reader will note that the objective function and the various constraints mimic in formal terms the scheduler definition given in section 4.1. In particular, the numbering of the constraints corresponds exactly. The extra generality in the definition of the objective function now realizes its intended payoff: By judicious choices of the parameters, the CHAMPS P&S can solve many different scheduling problems which might appear at first to be unrelated. For example, within the context of this formulation one could maximize the *value* of all jobs done, or maximize the *number* of jobs done. One could minimize *downtime*, or minimize the *costs associated with downtime*. (See [15] for simple techniques to estimate the cost of downtime.) One could minimize the *total execution time*. One could maximize the *number of jobs* which meet their deadlines. By employing a few additional tricks, such as the use of so called "dummy" tasks, one could minimize multiple deadline *penalties* associated with the jobs and/or tasks, for example those arising from customer *service level agreements (SLAs)* [3]. One could minimize the *average response time* or the *weighted average response time* of the various jobs.

### 4.3 Planner and Scheduler Example

Now we show an example of assigning tasks to servers using the RFC of installing the TPC-W `bestsell` servlet and other RFCs as jobs. There are 7 jobs and 5 servers. Servers 1 and 2 belong to one category (e.g., application servers) while servers 3, 4 and 5 belong to another category (e.g., database servers). Individual servers in the same category may have different CPU speeds, disk and memory resources.

Figure 4 shows the task to server assignments and schedules generated by the CHAMPS P&S for the 7 jobs. The $x$-axis represents time during a 6 hour change window, divided into 12 half hour intervals. All of the jobs are performed, and all of the constraints are met. For example, for Job 1, tasks 1, 2, 3, 4 and 19 are assigned to server 1, tasks 12, 13, 14, 15, 16, 17 and 18 are assigned to server 4, and tasks 5, 6, 7, 8, 9, 10 and 11 are assigned to server 5. These assignments meet the location-specific constraints. Notice that task 19 on server 1 does not start until task 8 on server 5 finishes, as dictated by the temporal constraints. A heavy retangular (possibly discontiguous) "box" is used to bound all

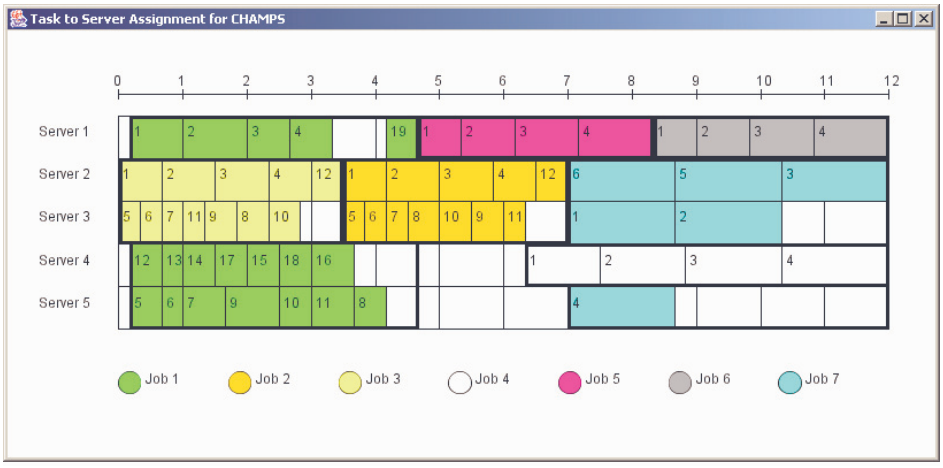


**Figure 4:** A Task Assignment Example

the tasks belonging to the same job. This is due to the decoupled nature of the CHAMPS P&S solution: The relevant servers are "reserved" for one job during the entire execution of that job, and thus blocked from any other assignments for the entire period inside the box.

## 5. Conclusions and Outlook

This paper describes the architecture and core concepts of the CHAMPS system, a prototype under development at IBM Research for **CHA**nge **M**anagement with **P**lanning and **S**cheduling. The CHAMPS system consists of a Task Graph Builder and a Planner & Scheduler. By decoupling Task Graph Building from Planning & Scheduling, and therefore binding Task Graphs to target systems at a fairly late stage in the process, we achieve a high degree of reusability for the information dealing with the software artifacts subject to a change. The optimization techniques we employ allow us to come up with a very hiqh quality solution for a mathematically intractable problem in a time which scales nicely with the problem size. In particular, the CHAMPS system is able to achieve a very high degree of parallelism for a set of tasks by exploiting factual knowledge about the structure of a distributed system from artifact dependency information at runtime. We have implemented the CHAMPS system and have applied it in a TPC-W environment that implements an on-line book store application.

While our initial results are encouraging, much work remains. We are currently working on scaling our approach to more complex multi-tiered application systems. On-line change plan adjustment, needed in case the roll-out of changes runs behind schedule, is being addressed by introducing a feedback mechanism from the deployment system into the Planner & Scheduler.

## References


[1] Business Process Execution Language for Web Services Version 1.1. Second Public Draft Release, BEA Systems, International Business Machines Corp., Microsoft Corp., SAP AG, Siebel Systems, May 2003. http://www-106.ibm.com/developerworks/library/ws-bpel/.

[2] A. Brown, G. Kar, and A. Keller. An Active Approach to Characterizing Dynamic Dependencies for Problem Determination in a Distributed Application Environment. In N. Anerousis, G. Pavlou, and A. Liotta, editors, *Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management*, Seattle, WA, USA, May 2001. IEEE Publishing.

[3] M. Buco, R. Chang, L. Luan, C. Ward, J. Wolf, and P. Yu. Managing eBusiness on Demand SLA Contracts in Business Terms. In *Proceedings of the 6th IFIP/IEEE International Symposium on Autonomous Decentralized Systems (ISADS 2003)*, Pisa, Italy, April 2003.

[4] M. Burgess. Cfengine: A site configuration engine. *Computing Systems*, 8(3), 1995. USENIX Association, see also: http://www.cfengine.org.

[5] Cisco Systems, Inc. *Change Management: Best Practices White Paper*, 2002.

[6] C. Ensel and A. Keller. An Approach for Managing Service Dependencies with XML and the Resource Description Framework. *Journal of Network and Systems Management, Special Issue: IM'2001 Selected Papers*, 10(2):147 – 170, June 2002.

[7] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, San Francisco, CA, USA, 1979.

[8] IBM Corporation, International Technical Support Organization. *All about IBM Tivoli Configuration Manager Version 4.2*, December 2002. IBM Redbook, Order Number: SG24-6612-00. see also: http://www.redbooks.ibm.com.

[9] Project Management Institute. *Guide to the Project Management Body of Knowledge*, 2000. see also: http://www.pmi.org.

[10] IT Infrastructure Library. *ITIL Service Support*, June 2000.

[11] F. Maurer and B. Dellen. Merging Project Planning and Web-Enabled Dynamic Workflow Technologies. *IEEE Internet Computing*, May 2000.

[12] *Microsoft Windows Update*. http://windowsupdate.microsoft.com.

[13] J.A. Nilsson and A.U. Ranerup. Elaborate change management: Improvisational introduction of groupware in public sector. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, 2001.

[14] D. Oppenheimer, A. Ganapathi, and D.A. Patterson. Why do internet services fail, and what can be done about it? In *Proceedings of the 4th Usenix Symposium on Internet Technologies and Systems*, Seattle, WA, USA, March 2003. USENIX Association.

[15] D. Patterson. A Simple Way to Estimate the Cost of Downtime. In *Proceedings of the Sixteenth Systems Administration Conference (LISA 2002)*, Philadelphia, PA, USA, November 2002. USENIX Association.

[16] D. Ressman and J. Valdes. Use of Cfengine for Automated, Multi-Platform Software and Patch Distribution. In *Proceedings of the Fourteenth Systems Administration Conference (LISA 2000)*, New Orleans, LA, USA, December 2000. USENIX Association.

[17] Transaction Processing Performance Council. *TPC Benchmark W Specification (Web Commerce) v1.8*, February 2002. http://www.tpc.org/tpcw.

[18] S. Traugott and J. Huddleston. Bootstrapping an Infrastructure. In *Proceedings of the Twelfth Systems Administration Conference (LISA 98)*, Boston, MA, USA, December 1998. USENIX Association. see also http://www.infrastructures.org/.

[19] A. van Hoff, H. Partovi, and T. Thai. The Open Software Description Format (OSD). August 1997. http://www.w3.org/TR/NOTE-OSD.html/.