Optimal Resource-Aware Deployment Planning for Component-based Distributed Applications

Tatiana Kichkaylo and Vijay Karamcheti
New York University
New York, NY 10012, USA
{*kichkay, vijayk*}*@cs.nyu.edu*

Abstract

Component-based approaches are becoming increasingly popular in the areas of adaptive distributed systems, web services, and grid computing. In each case, the underlying infrastructure needs to address a deployment problem involving the placement of application components onto computational, data, and network resources across a wide-area environment subject to a variety of qualitative and quantitative constraints. In general, the deployment needs to also introduce auxiliary components (e.g., to compress/decompress data, or invoke GridFTP sessions to make data available at a remote site), and reuse pre-existing components and data. To provide the flexibility required in the latter case, recently proposed systems such as Sekitei and Pegasus have proposed solutions that rely upon AI planning-based techniques.

Although promising, the inherent complexity of AI planning and the fact that constraints governing component deployment often involve non-linear and non-reversible functions have prevented such solutions from generating deployments in resource-constrained situations and achieving optimality in terms of overall resource usage or other cost metrics. This paper addresses both of these shortcomings in the context of the Sekitei system. Our extension relies upon information supplied by a domain expert, which classifies component behavior into a discrete set of levels. This discretization, often justified in practice, permits the planner to identify cost-optimal plans (whose quality improves with the level definitions) without restricting the form of the constraint functions. We describe the modified Sekitei algorithm, and characterize, using a media stream delivery application, its scaling behavior when generating optimal deployments for various network configurations.

1 Introduction

A growing number of distributed applications spanning varied areas such as adaptive component frameworks, web services, and grid computing, are being structured as aggregations of multiple independent components. Components cooperate to realize application functionality by invoking each other's services, processing data streams, or reading and writing files. Given the modularity benefits of well-defined component interfaces, the notion of a "distributed application" is shifting from the traditional view of statically deployed entities into one defined by a high-level description of its components, their locations, and the linkages between them. Different domains use different terminology and representations for this high-level description. For instance, current-day grid applications rely upon scripted interactions between logical resources, which are executed by tools such as Condor DAGman [16]. In the web services area, an application is represented by a BPEL or OWL-S composite service [1, 2]. Similarly, adaptation-capable component frameworks such as Partitionable Services [8], describe the application in terms of type-based linkages between component interfaces.

Despite different representations, the underlying infrastructure in each case needs to solve a common deployment problem that we refer to as the *component placement problem* (CPP). In general, CPP involves the placement of application components onto computational, data, and network resources across a wide-area environment subject to a variety of qualitative and quantitative constraints. For instance, in a grid computing application described in terms of a task graph exchanging information using logical files [3], a solution to the CPP would result in a mapping of tasks to concrete components on specific computational hosts, the mapping of logical files to physical replicas, and orchestration of any required data transfers across the different hosts. Thus, the solution needs to satisfy both logical (*qualitative*) constraints on the components (e.g., that a certain kind of task operates on a certain logical file) as well as resource-oriented (*quantitative*) constraints (e.g., that data transfers would consume no more than a specified amount of CPU or network resources). Ideally, the solution should also achieve optimality with respect to one or more metrics,

e.g., consume the least amount of resources and/or involve the fewest number of components.

The general nature of CPP requirements — (1) the need to satisfy both qualitative and quantitative constraints, and (2) the fact that application deployment may involve choosing amongst compatible components as well as insertion of auxiliary components — means that the CPP cannot be solved by just optimizing the mapping of components onto network resources. For this reason, recently proposed systems such as Pegasus [3] and Sekitei [11] have advocated the use of general AI planning approaches.

Although promising, such approaches suffer from an inherent conflict between capturing general constraints and achieving optimality. To clarify this using an example, in our Sekitei system targeting the Partitionable Services framework [11, 8], general constraints (both on interface properties and component resource consumption) are represented using non-reversible functions involving real-valued variables. The implication of non-reversibility is that the planner needs to adopt a greedy "worst-case" approach when allocating network resources. While sufficient for several situations, the greedy approach suffers from two shortcomings. First, it may, in resource-constrained situations, not produce a deployment plan when one does in fact exist. Second, the approach is unable to achieve optimality in terms of overall resource usage or other user-specified cost metrics. Both shortcomings stem from the inability of the approach to reason about minimal requirements on resource consumption (more generally, constraint satisfaction) in the presence of non-reversible functions.

This paper describes our extensions to the Sekitei model and algorithm for addressing these shortcomings. Although one could restrict the constraint functions so that they are reversible, this restriction is incompatible with the behavior of real components, typically captured as a table of profiled values. We adopt a different approach that retains function generality, but overcomes the reasoning obstacle by relying upon information supplied by a domain expert, which classifies component behavior into a discrete set of levels. This discretization, which is practically justified (we often think of components as behaving in different operation regimes), permits the planner to identify cost-optimal plans addressing both of the shortcomings above. The quality of the generated plan depends on the level definitions, but as our experimental results show, substantial benefits can be obtained even with relatively imprecise definitions. In terms of practical impact, the planner is better able to generate plans in resource-constrained environments and achieve user-specified notions of optimality. For example, the modified Sekitei planner is capable of deploying the task graph scenario described earlier in a way that minimizes resource consumption while meeting specified deadline goals.

We start, in Section 2, by presenting the component placement problem encountered in Partitionable Services [8], and review the basic Sekitei algorithm [11]. Section 3 presents the main ideas of the paper, describing levels and a modified Sekitei algorithm that takes advantage of them. For clarity reasons, we restrict our attention to constraints governing resource consumption (e.g., of node CPU and link bandwidth), but note that our ideas are easily extendible to general constraints and other CPP formulations. Section 4 evaluates the scaling behavior and solution quality of the modified algorithm using a media stream delivery application. Section 5 discusses related work, and we conclude in Section 6.

2 The Component Placement Problem in Partitionable Services

Partitionable Services [8] is an example of dynamic component-based frameworks [6, 7], which permit distributed applications to adapt to their execution environments by dynamically selecting, composing, and mapping their constituent components. Applications in the framework are described at a high-level as type-compatible aggregations of components, whose functionality is expressed in terms of well-defined interfaces. We restrict our attention in this paper to a special case of such applications, where components consume and produce data streams that are sent across links between nodes in a wide-area network.

Figure 1 illustrates one such application. The ***Server*** component, running on node 7, provides a combined media (M) stream consisting of images and text, which must be received with a certain minimum bandwidth by the ***Client*** component on node 0. If there is a network path of sufficient bandwidth between the two nodes, a direct connection can be established. Otherwise, the data stream needs to be transformed. ***Splitter*** and ***Merger*** components can be used to divide the M stream into its text (T) and image (I) components to enable their transmission along different paths, with compression components (***Zip*** and ***Unzip***) providing additional bandwidth reduction for the T stream.

The CPP for this application would, given as input a model of network resources and the application structure, determine which of the above components are required, where they are placed, and how they are linked together so as to satisfy client bandwidth requirements.

2.1 The basic model

The CPP is specified by a network topology and resources, specifications of components, and a characterization of the interactions between components and the network environment [11].

The network is assumed built up out of nodes and links, each characterized in terms of a number of resources. Our

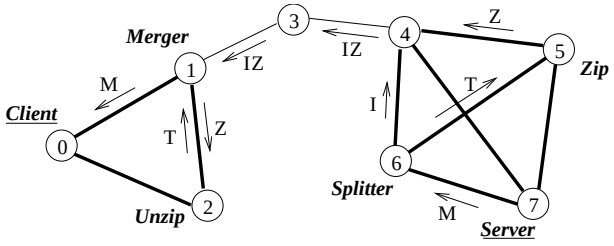

Figure 1. The example application. To deliver the M stream from the server node to the client node, additional components are injected into the data path to transform the data stream in order to cope with low bandwidth of the path between nodes 1 and 4.

resources of interest include node CPU and link bandwidth. In general, additional resources such as node memory and disk bandwidth may be relevant, as well as other properties such as link security, the available software on a node, etc.

A component is defined as consuming and producing zero or more interfaces (data streams), each of which is associated with a number of application-specific properties. The interfaces in our example have an `ibw` property, which corresponds to the stream bandwidth. The component specification additionally contains formulae describing resource requirements and effects of component deployment. For example, the merger component described in Figure 2 requires text (T) and image (I) streams and produces a combined media (M) stream. This component can be deployed on a node if there are sufficient CPU resources, and can process the incoming streams when their bandwidths (rates) are in a particular relation to each other. As a result of a deployment, node CPU resources are consumed, and a new M stream is generated with bandwidth defined as a function of the incoming streams.

Similar to the effect formulae, specification of each interface contains formulae describing the interactions of that interface with network links. This might include consumption of link bandwidth, accumulation of latency on the stream, or a change in application-specific properties such as the delivered stream bandwidth.

Note that in general, the formulae referred to above are *non-reversible* functions of real-valued resource and property variables. Non-reversibility refers to the fact that for the specification above, it is *not* possible to compute the required resources and properties on all input streams given properties of any of the output streams. For example, in Figure 2, given a value for the `ibw` property of the output M stream, in general, there is no way of computing the required node CPU resources and values of the `ibw` property of the input T and I streams.

<component name=*Merger*>
  <linkages>
    <requires>
      <interface name=T>
      <interface name=I>
    <implements>
      <interface name=M>
  <conditions>
    Node.cpu >= (T.ibw+I.ibw)/5
    T.ibw*3 == I.ibw*7
  <effects>
    M.ibw := T.ibw + I.ibw
    Node.cpu -= (T.ibw+I.ibw)/5

Figure 2. Component specification.

2.2 The Sekitei planner

In [11, 12] we described an algorithm, called Sekitei, which solves the CPP by viewing it as an AI-style planning problem. The latter has two types of actions: (1) placement of a component on a node (`placeX(?node)`, where `X` is the name of the component, and `?node` is a variable describing the node); and (2) the crossing of a network link with an interface (`cross(?interface ?fromNode ?toNode)`). Sekitei determines the sequence of actions that produce the desired goal (availability of certain interfaces on certain nodes). Since the scale considerations of the CPP are very different from classical AI planning domains, Sekitei incorporates a number of domain-specific optimizations [12].

Sekitei deals with the non-reversible nature of functions in the CPP specification by assuming that the functions are monotonic (e.g., that if the bandwidth of the I input stream in Figure 2 increases, that the bandwidth of the output M stream would not decrease), and adopting a greedy approach while planning by considering the maximum possible utilization of a resource. The rationale for the latter is that if a plan is feasible assuming maximum amount of data being pushed through the deployed components and network resources, it continues to be feasible even when lower amounts of data are transfered.

2.3 Shortcomings of the basic algorithm

A consequence of the greedy approach described above is that Sekitei-like planning approaches can guarantee feasibility, but cannot minimize resource consumption. In our original version, a post-processing step attempted to achieve this latter goal, but this is not enough as the following examples demonstrate:

Scenario 1. Consider the example in Figure 3, where we want to deliver at least 90 units of bandwidth of the M

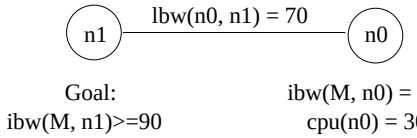

Figure 3. Resource optimization is required to find a plan.

```
place Splitter on node n0,
place Zip on node n0,
cross with Z stream from n0 to n1,
cross with I stream from n0 to n1,
place Unzip on node n1,
place Merger on node n1.
```

Figure 4. Plan for the problem presented in Figure 3.

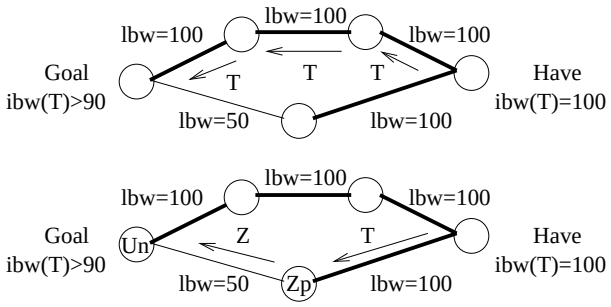

Figure 5. Effect of cost functions on the choice of plan.

stream (the requirement of the client component) over the link with bandwidth 70. The source node has 200 units of M available, but only 30 units of CPU. Suppose, transformation of 200 units of M by the splitter requires 40 units of CPU. Sending the M stream directly to the client does not satisfy client's bandwidth requirements, and the amount of CPU available on node `n0` is less than that required for processing all available bandwidth of the M stream, as would be required by the greedy approach. Consequently, the latter will not find a solution to the CPP even though one exists. If we allow the splitter to transform only 90 units of bandwidth of the available M stream (the amount required by the client), then the total CPU requirements of the Splitter and Zip components may be less than 30 units, and the solution shown on Figure 4 can be found, which involves splitting the M stream and compressing its text component on node `n0` and performing the reverse transformations on node `n1`.[1]

[1]We assume that the target node has sufficient CPU resources for the Unzip and Merger components.

Scenario 2. Another desirable feature not provided by the basic model is the ability to specify preferences over the space of generated plans. For example, in the scenario in Figure 5, there are two possible ways to deliver sufficient bandwidth of the T stream from the server node to the client: one involving a crossing of three links, and another that would require two link crossings and the use of Zip and Unzip components. Which plan would perform better in a given situation depends on the relative cost of link bandwidth and node resources. Such tradeoffs can be performed by introducing a cost function that depends on resource consumption, which an ideal planner can then optimize. Note that, in general, the cheapest plan is not necessarily the one with the smallest number of steps.

3 Resource-Optimal Solutions for the CPP

The scenarios described in the previous section represent two general shortcomings of the greedy planning approach adopted in Sekitei-like planners: inability to find plans in resource-constrained environments, and inability to optimize resource consumption or other user-supplied cost and performance metrics. The fundamental reason underlying these shortcomings is the non-reversible nature of the resource functions, which prevent the planner from reasoning about resource availability and consumption.

Thus, to address these issues, one needs a way of permitting the planner to understand what effect its actions have upon network resources. The simplest way of doing this is to assume reversibility of functions; however, this is at odds with what one finds in practice. Functions describing component behavior are often represented by tables obtained by application profiling. It is not always possible to derive an analytical representation of such functions, and even less reasonable to assume reversibility of such functions. Consequently, we adopt a different approach, which approximates optimality while still being practically usable. We start by providing the intuition behind our approach, and then describe a modified algorithm to solve the CPP.

3.1 Intuition: Resource Levels, Leveled Actions

The key insight underlying our approach is that more than an exact understanding of the resource consumption effects of a planning action, what we care about is the ability to identify actions that come close to the right (optimal) decision. The latter is somewhat easier and more reasonable for a domain expert to provide information on. In particular, we leverage the observation that experts are already used to thinking of different operational regimes for components as also qualitatively different regions of values for network resources, to augment the basic planning problem defined in Section 2 with the notion of *resource levels*.

```
<interface name=M>
 <cross_effects>
    M.ibw'   := min( M.ibw, Link.lbw )
    Link.lbw' -= min( M.ibw, Link.lbw )
 <levels>
    <cutpoint value=30>
    <cutpoint value=70>
    <cutpoint value=90>
    <cutpoint value=100>
```

Figure 6. Specification of an interface with resource levels. The tick mark in the specifications serves to distinguish the value of a resource after the link crossing operation.

Resource levels Every interface property or network resource, which appears as a real-valued variable in a specification formula, is assumed to have one or more **levels** associated with it. The levels specify disjoint intervals of values of the resource and are defined by the interval bounds. Resources for which no intervals are specified are assumed to have one interval $[0, \infty)$. For example, the specification of the M stream shown in Figure 6 defines five intervals for the bandwidth property: $[0, 30)$, $[30, 70)$, $[70, 90)$, $[90, 100)$, and $[100, \infty)$.

Additionally, a property or a resource can be marked as being *degradable*, *upgradable*, or neither. A degradable resource tag indicates that the availability of a resource at a higher value indicates its availability at a lower value as well. For example, link bandwidth is a degradable resource. Similarly, an upgradable resource is assumed available at a higher value when a lower value is present. Information about degradability (upgradability), which can be obtained automatically by syntactic analysis of the problem specification or provided manually, helps the planner to find plans in resource-constrained situations as described below.

Leveled actions The main benefit from identifying resource levels is that we can incorporate that information when defining actions for the AI-style planning problem compiled from the CPP specification. Specifically, we introduce two extensions to the basic model described earlier. First, levels for all resources mentioned in the action specification are added as parameters to the action. Whenever possible, additional (in)equalities can also be added to action preconditions to limit possible combinations of level values. Second, an action is extended with a user-specified cost formula dependent on the values of its resource variables. For example, the cost of placing a Merger component on a network node might be defined as a function of the total processed bandwidth: `1+(I.ibw+T.ibw)/10`.

Since the planning algorithm can be thought of as performing directed search through the space of possible action sequences, the implication of the above two extensions is that this search process can be guided by a more detailed knowledge of resource utilization and plan cost metrics than possible in the original model. Specifically, the level information helps prune out certain actions during the planning process by dictating the resource assumptions that must be satisfied for an action to be selected. Formally, the intervals corresponding to the resource levels form an action's **optimistic resource map**. The fact that the optimistic map of an action `ac` contains an interval `[m, M)` for a resource variable `X` implies that `ac` has a resource precondition $\texttt{m} \leq \texttt{X} < \texttt{M}$, and this information can be used for pruning. Optimistic maps also provide benefits for estimating the cost of performing an action sequence, which enables the use of A*-like search strategies to optimize the plan cost.

The level information is used along with the notions of degradability and upgradability defined earlier to guide the search procedure to explore other alternatives when the current path does not yield a solution. This feature is what allows the planning algorithm we describe below to come up with a plan for Scenario 1 that transmits a lower amount of M stream bandwidth upon finding that network resources are inadequate for the maximum amount.

3.2 Algorithm

After compilation and leveling, the CPP is described by a set of AI-style planning actions, each specified using propositional as well as non-reversible real-valued preconditions and effects. The planning algorithm is guided primarily by the propositional part, and uses the real-valued functions for pruning and ensuring soundness of the solution.

The algorithm proceeds in phases, using solutions to relaxed problems to estimate the cost of achieving goals. The first phase of the algorithm estimates the minimum cost of achieving a proposition from the initial state. Both resource restrictions and interactions between actions are ignored in this case.[2] Given the minimum proposition cost, the second phase computes the minimum logical cost of achieving a *set* of propositions. This phase takes into account logical interactions between actions, but ignores resource restrictions. Finally, during the last phase, the search for a plan is performed that uses all types of restrictions and estimates the remaining cost using the logical cost of achieving a set of propositions.

3.2.1 Cost of propositions

The algorithm first constructs a per-proposition logical regression graph (PLRG), which estimates the minimum logical cost of achieving a proposition from the initial state and identifies the set of relevant actions [11]. Since the PLRG

[2] Except for the resource restrictions reflected by the leveling of actions.

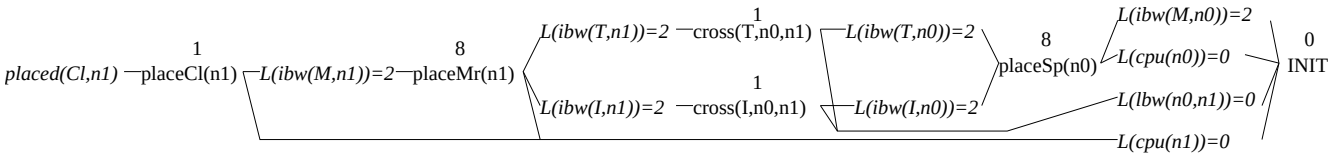

Figure 7. A part of the PLRG for the problem shown on Figure 3. The notation *L(v)=n* means that the resource variable `v` has level `n`. Numbers above action nodes show costs of those actions given the resource levels. `Cl` stands for Client, `Sp` for Splitter, and `Mr` for Merger.

only considers logical preconditions and effects, its cost estimates are a lower bound on the actual cost of achieving a proposition, and therefore can be used as an admissible heuristic in the later stages of the algorithm.

The PLRG is expanded from the goal state until a solution is obtained, a bound is reached, or no further expansion is possible. The latter implies that the goal is logically unreachable from the initial state, and that the problem has no solution.

Figure 7 shows a portion of the PLRG for the problem in Figure 3. Actions for crossing the link with the M stream with levels above 1 are pruned during the leveling because of limited link bandwidth. Therefore, the cheapest way to achieve the proposition `L(ibw(M,n1))=2`, which states that the M stream bandwidth on node `n1` is in the second level interval, is to use Splitter and Merger components.

The PLRG consists of action and proposition nodes, and thus contains information about logical support. When estimating the cost of a proposition, the cost of a proposition node is taken as the minimum of the costs of supporting actions, and the cost of an action node as the maximum cost of its preconditions. For example, the logical cost of achieving the proposition `placed(Cl,n1)` in Figure 7 is 18. Obtaining this cost requires sending both image and uncompressed text streams over the link. This would lead to violation of client's bandwidth requirements, but this fact cannot be detected in the PLRG.

3.2.2 Cost of sets of propositions

The second phase of the algorithm estimates the minimum logical cost of a set of propositions using the Set LRG (SLRG). The nodes of the SLRG correspond to sets of propositions. New nodes are generated by regressing over actions. The construction of the SLRG employs A* search and uses the logical cost of achieving propositions obtained from the PLRG as an estimate of the remaining cost.

The estimate of the cost of a set of propositions by the SLRG is more accurate than that obtained directly from the PLRG. For example, the cost of achieving a singleton set {`placed(Cl,n1)`} is 19, because the two link crossing actions `cross(T,n0,n1)` and `cross(I,n0,n1)` are now considered in sequence rather than in parallel.

During the leveling of actions, the level propositions are created for all resource variables mentioned in actions. However, only levels of interfaces need to be *achieved*, and the rest are only checked. Only actions that achieve such **important** propositions are used for branching.

The SLRG computes set costs for important propositions only. For each important proposition, the best achievable levels of unimportant resources are computed in the PLRG. This information is then used to improve estimates of the cost of achieving sets of important propositions. For example, if both interface bandwidth and link bandwidth are leveled, it may be possible to detect in the SLRG the fact that sufficient levels of text and image streams cannot be delivered by using `cross(T,n0,n1)` and `cross(I,n0,n1)` actions, because both of them also decrease the level of available link bandwidth.

3.2.3 The main regression graph

The final phase of the algorithm is construction of the main regression graph (RG). The RG contains totally ordered plan tails and is expanded using A* search. The logical cost of achieving a set of propositions is used as an estimate of the remaining cost.

Whenever a new node is created by regressing the current cheapest node over an action, the plan tail including this action is replayed in the optimistic map of this action (see Figure 8). If the execution fails, the new node is pruned from the search. Such partial execution allows early detection of violations of quality-of-service requirements, and, for example, discarding of partial plans whose total latency exceeds a given limit.

The optimistic map contains intervals for all resource variables required by the action as specified by its leveled resource preconditions. Before execution of each subsequent action in the plan tail, the interval produced by execution of the previous action is intersected with the optimistic interval of the current action, and new optimistic intervals are added if necessary (Figure 8).

The main difference between SLRG and RG is propagation of resource maps in the RG. Since resource failures

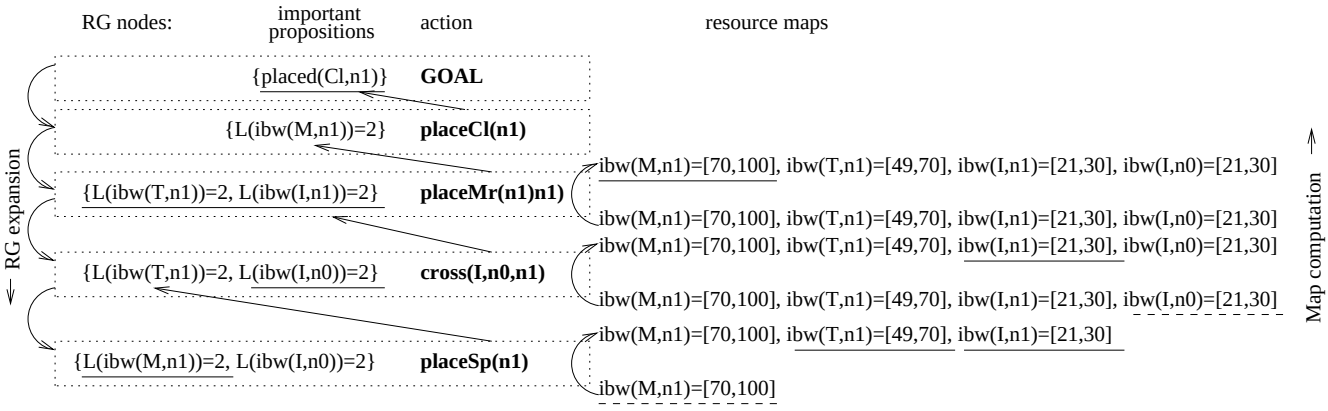

Figure 8. Propagation of resource maps in the RG. Each RG node has an action and a set of propositions describing the state in which the action is to be executed (only important propositions are shown). The action of a node needs to achieve at least one proposition of the parent node. Arrows connect actions to propositions they achieve. Logical preconditions of actions are underlined. In the resource maps, dashed lines mark newly added optimistic intervals, and solid lines show values added as a result of action execution. Intervals resulting from action execution are merged with the previous values for the same resource.

depend on the plan tail, it is not possible to reuse nodes in the RG. The RG is a tree, while the PLRG and SLRG are general graphs.

The search in the RG ends when all propositions, both important and unimportant, are present in the initial state, and the plan tail successfully executes in the resource map of the initial state.

4 Evaluation and Discussion

Extending the basic model of the CPP with cost functions and resource levels pursues two goals: allowing the planner to find solutions in resource constrained situations (Scenario 1) and specifying preferences over plans (Scenario 2). We decided to achieve this functionality by optimizing a cost function depending on resource consumption. Given approximation of actual resource values by discrete levels, our algorithm optimizes the minimum cost of the plan instead of the exact cost. However, in our examples this approximation was sufficient.[3]

The ability of our planner to achieve the desired functionality depends greatly on the actual specification of levels. Without levels, or with a poor choice of values for levels, the benefits from additional functionality are lost (however, solutions found by the planner are still correct). On the other hand, using multiple levels for each resource increases the size of the problem and negatively affects performance of the planner.

The following experiment shows how the choice of levels affects scalability of the planner and quality of solutions. Since the focus of this paper is on the ability of AI planning approaches to generate resource-aware plans (assuming that the specifications provided to it are correct), we evaluate the algorithm using simulated network configurations.

4.1 The experiment

We tested the planner on the CPP described in Figure 1 with three different sizes of the network and five different level specifications.

The CPP involves delivering a media stream from the server to the client. Locations of both the server and the clients are given. The client requires at least 90 units of bandwidth of the media stream, and the server is capable of producing up to 200 units. The costs of component placement and link crossing are proportional to the processed/transfered bandwidth. Such definition of the cost favors application configurations with the minimum number of additional components and the minimum bandwidth consumption along the data path.

The three networks used in our experiments have the same distribution of resources. LAN links of the networks have bandwidth 150 units, WAN links 70 units. The CPU resources on all nodes are sufficient for placing Splitter and Zip (or Unzip and Merger) components to process up to 111 units of the media stream. The *Tiny* scenario corre-

[3] With some extra effort it is possible to ensure that the real cost of the plan, rather than the lower bound, is optimal. However, as we discuss below, this is usually unnecessary.

Scenario	Levels of bandwidth of M	Levels of link bandwidth
A	$[0, \infty)$	$[0, \infty)$
B	$[0, 100)$, $[100, \infty)$	$[0, \infty)$
C	$[0, 90)$, $[90, 100)$, $[100, \infty)$	$[0, \infty)$
D	$[0, 30)$, $[30, 70)$, $[70, 90)$, $[90, 100)$, $[100, \infty)$	$[0, \infty)$
E	$[0, 30)$, $[30, 70)$, $[70, 90)$, $[90, 100)$, $[100, \infty)$	$[0, 31)$, $[31, 62)$, $[62, \infty)$

Table 1. Resource level scenarios. Bandwidth levels of interfaces T, I, and Z are proportional to those of the M stream.

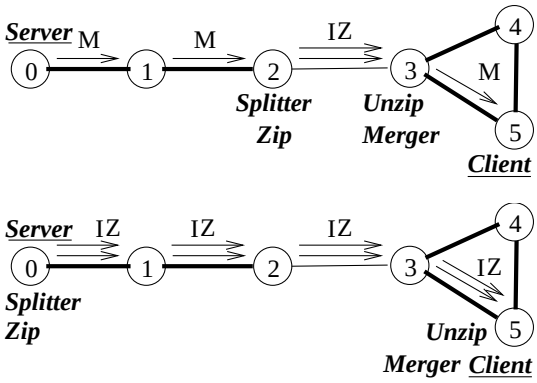

Figure 9. Suboptimal and optimal plans for the *Small* network.

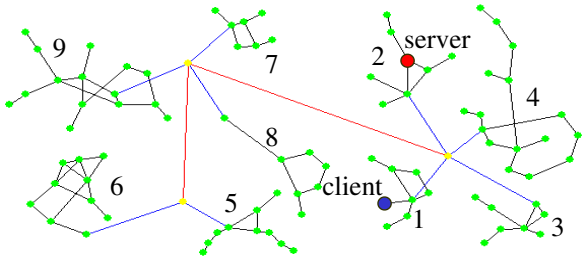

Figure 10. The 93-node network for the *Large* scenario.

sponds to the two-node network shown in Figure 3. Given any reasonable cost function, the plan in this case contains 7 actions (the six actions shown on Figure 4 plus the client placement). The *Small* scenario involves a 6-node network. The shortest plan has 10 actions and cost of 72 (Figure 9 top). Since the media stream is sent over the LAN links, the bandwidth required there is 90 units. The optimal plan has 13 actions and a cost of 63 (Figure 9 bottom). This plan requires only 27+31.5=58.5 units of bandwidth of LAN links. Finally, the *Large* scenario corresponds to the similar problem in a 93-node network (Figure 10) generated using the GeorgiaTech ITM tool [18]. Most of the nodes of this network do not participate in the plan, but cannot be statically pruned.

Table 1 shows the five resource scenarios. Scenario A corresponds to the original version of Sekitei (without resource levels). In this scenario, the limited network resources prevent the planner from finding any plan. Table 2 shows experimental results for the other 4 scenarios on each of the three network configurations.

4.2 Quality of solution

Even a single cut point 100 introduced in scenario B, which puts an upper bound on resource consumption, allows the planner to find a solution whiere only 100 units of the available stream are processed. However, in the *Small* and *Large* networks the found application configuration is suboptimal with respect to the reserved LAN link bandwidth (Column 4).

To ensure that the found plan is optimal with respect to the user-specified cost function, the lower bound on the cost function (Column 2 in the table) obtained by the planner needs to approximate the real cost of the plan as close as possible. The level specifications of scenarios C, D, and E allow the planner to select the best configuration.

The plans selected in scenarios C, D, and E involve processing 100 units of bandwidth of the M stream, which is more than strictly required to satisfy the client's requirements. The best quality of a solution would be achieved if the bandwidth of the media stream is cut at two points exactly around 90. Obtaining such values automatically requires reversibility of resource functions. Scenario C approximates the ideal values: It selects the optimal configuration, but requires slightly more resources than absolutely necessary (the bandwidth required on LAN links is 65 instead of the optimal 58.5).

4.3 Scalability

Table 2 also provides information on scalability of our algorithm. The first number in Column 9 is the total time including reading problem files and construction of actions. The second number is the time spent in search and construction of the graphs. Column 5 gives the total number of actions evaluated after leveling and using the pruning procedure. Sizes of the three graphs characterize memory re-

Scenario		*Quality of the solution*			*Work done by the planner*				
		lower bound on cost	actions in plan	reserved LAN bw	total # of actions	graph sizes			planning time (ms)
						PLRG	SLRG	RG	
1		2	3	4	5	6	7	8	9
Tiny	B	7	7	N/A	32	27 / 19	24	25 / 7	260 / 90
	C	42	7	N/A	46	26 / 20	24	16 / 5	271 / 70
	D	42	7	N/A	76	26 / 22	24	16 / 5	310 / 60
	E	42	7	N/A	174	48 / 25	51	28 / 16	331 / 70
Small	B	10	10	100	152	138 / 49	139	246 / 159	721 / 420
	C	63	13	65	222	136 / 50	188	98 / 64	611 / 291
	D	63	13	65	364	136 / 52	188	98 / 64	731 / 330
	E	63	13	65	1152	366 / 70	2888	3198 / 2558	5128 / 4366
Large	B	11	11	100	2582	2286 / 742	3278	2348 / 1949	12418 / 3205
	C	63	13	65	3780	2368 / 746	1062	254 / 203	10405 / 1041
	D	63	13	65	6198	2368 / 748	1062	216 / 176	12138 / 671
	E	63	13	65	20386	6594 / 1066	76179	4557 / 4243	40077 / 25426

Table 2. Scalability evaluation.

quirements of the planner. The table gives the number of proposition and action nodes for the PLRG, the total number of set nodes for the SLRG, the total number of RG nodes, and the number of RG nodes left in the A* queue at the moment when a solution is found.

As the results show, introduction of resource levels significantly increases the number of generated actions. However, it also permits identification of some resource conflicts at earlier (and cheaper) phases of the search, which explains the improved performance of Scenario C compared to Scenario B and in several cases that of scenario D over C.

Adding more levels of interface bandwidth (scenario D) and leveling link bandwidth (scenario E) does not always improve the quality of solution, but negatively affects performance of the planner. The good choice of levels depends on requirements of application components and on the definition of the cost function. Although it is not the case in the presented experiment, we expect that for some problems it might be beneficial to discretize such resources as link bandwidth and node CPU. In the presence of non-reversible resource functions the choice of levels needs to be performed by a domain expert, possibly, based on profiling results.

5 Related Work

The general CPP is at least PSPACE-hard [10], and existing planners usually restrict themselves to special cases of the general problem. The CANS planner [5] can find optimal deployments of chains of components along network paths. The Ninja planner [7] constructs DAG-structured applications out of components already available in the network. Pegasus [3] is a planning architecture for construction of grid applications. Pegasus heavily relies on external services for choosing components and nodes and does not explicitly reason about resources. Sekitei can employ external services, e.g., for enforcing matchmaking policies [14], by incorporating calls to such services into the resource functions. Recall that the only restriction on such functions is monotonicity.

Planning with real-valued resources has also been investigated by the AI community. SAPA [4] performs forward search, and therefore suffers from the same problems as the greedy version of Sekitei. Compilation-based planners achieve good performance by using fast algorithms for solving a satisfiability or optimization problem constructed from the original planning problem. Such planners also provide additional functionality by supporting explicit optimization [9], and can, in principle, be used to optimize resource consumption. However, such planners are limited to linear functions and do not scale well with the size of a problem specification. [17] and [15] propose to perform resource selection (scheduling) separately from action selection (planning). Such an approach works well when the two problems are loosely coupled. In the CPP, action selection is driven by the resource restrictions. Without the latter, there is usually a trivial solution to the problem (e.g. to connect the client and the server directly in our example).

6 Summary

In this paper, we have presented an extended model of the component placement problem that allows for explicit optimization of resource consumption of generated deployment plans and reasoning about plan costs. This extensions allows the planner to find a solution in some resource con-

strained situations where the traditional approach fails. Optimizing the cost function also allows tradeoffs between different resources depending on their costs. For example, one can choose between longer network paths and additional computation.

To construct an efficient planner, we chose to use discrete resource levels instead of continuous resource variables in action parameters. Such an approximation is sufficient to achieve the desired functionality given a good choice of level boundaries. We presented a planning algorithm that works with this model. Our preliminary experiments with the component placement problem show that the algorithm achieves good performance despite the increase in the number of actions. An alternative approach might be to use optimization techniques to solve the problem with continuous variables. We expect such approaches to have significantly worse performance compared to our algorithm without providing sufficient improvement in the quality of solutions.

In the future, we plan to analyze the dependency between the number and quality of resource levels and performance of the algorithm in hopes of improving the algorithm's performance and scalability. We also intend to use our planner for repairing and adapting existing deployments by introducing operators for migrating and reconnecting components. Separate operators are necessary, because the cost of migration differs from that of the initial deployment.

7 Acknowledgments

This research was sponsored by DARPA agreements N66001-00-1-8920 and N66001-01-1-8929; and by NSF grants CAREER:CCR-9876128 and CCR-0312956. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of DARPA, SPAWAR SYSCEN, or the U.S. Government.

References

[1] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services. http://www-106.ibm.com/developerworks/library/ws-bpel/, 2003.

[2] A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, and H. Zeng. DAML-S: Semantic markup for web services. In *Semantic Web Working Symposium*, 2001.

[3] J. Blythe, E. Deelman, Y. Gil, C. Kesselman, A. Agarwal, G. Mehta, and K. Vahi. The role of planning in grid computing. In *ICAPS*, 2003.

[4] M. Do and S. Kambhampati. Sapa: A scalable multi-objective metric temporal planner. *JAIR*, 2003.

[5] X. Fu and V. Karamcheti. Planning for network-aware paths. In *DAIS*, 2003.

[6] X. Fu, W. Shi, A. Akkerman, and V. Karamcheti. CANS: Composable, Adaptive Network Services infrastructure. In *USITS*, 2001.

[7] S. Gribble, M. Welsh, R. von Behren, E. Brewer, D. Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R. Katz, Z. M. Mao, S. Ross, and B. Zhao. The Ninja architecture for robust Internet-scale systems and services. *Computer Networks*, 35(4):473–497, 2001.

[8] A.-A. Ivan, J. Harman, M. Allen, and V. Karamcheti. Partitionable services: A framework for seamlessly adapting distributed applications to heterogenous environments. In *HPDC*, 2002.

[9] H. Kautz and J. Walser. State-space planning by integer optimization. In *AAAI*, 1999.

[10] T. Kichkaylo. Timeless planning and the component placement problem. In *ICAPS Workshop on Planning and Scheduling for Web and Grid Services*, 2004.

[11] T. Kichkaylo, A. Ivan, and V. Karamcheti. Constrained component deployment in wide-area networks using AI planning techniques. In *IPDPS*, 2003.

[12] T. Kichkaylo, A. Ivan, and V. Karamcheti. Sekitei: An AI planner for constrained component deployment in wide-area networks. Technical Report TR2004-851, New York University, 2004.

[13] J. Koehler and J. Hoffmann. Handling of inertia in a planning system. Technical Report 122, Albert-Ludwigs-University, 1999.

[14] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *HPDC*, 1998.

[15] B. Srivastava. Realplan: Decoupling causal and resource reasoning in planning. In *AAAI*, 2000.

[16] D. Thain, T. Tannenbaum, and M. Livny. Condor and the grid. In *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., 2002.

[17] S. Wolfman and D. Weld. Combining linear programming and satisfiability solving for resource planning. *The Knowledge Engineering Review*, 2001.

[18] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *IEEE Infocom*, volume 2, 1996.