Bandwidth-Limited Optimal Deployment of Eventually-Serializable Data Services

Laurent Michel[2], Pascal Van Hentenryck[1], Elaine Sonderegger[2], Alexander Shvartsman[2], and Martijn Moraal[2]

[1] Brown University, Box 1910, Providence, RI 02912
[2] University of Connecticut, Storrs, CT 06269-2155

Abstract. Providing consistent and fault-tolerant distributed object services is among the fundamental problems in distributed computing. To achieve fault-tolerance and to increase throughput, objects are replicated at different networked nodes. However, replication induces significant communication costs to maintain replica consistency. Eventually-Serializable Data Service (ESDS) has been proposed to reduce these costs and enable fast operations on data, while still providing guarantees that the replicated data will eventually be consistent. This paper revisits ESDS instances where bandwidth constraints are imposed on segments of the network interconnect. This class of problems was shown to be extremely challenging for both Mixed Integer Programming (MIP) and for Constraint Programming (CP), some instances requiring hours of computation time. The paper presents an improved constraint programming model, a constraint-based local search model that can obtain high-quality solutions quickly and a local search/constraint programming hybrid. The experimental results indicate that the resulting models significantly improve the state of the art.

1 Introduction

Data replication is a fundamental technique in distributed systems: it improves availability, increases throughput, and eliminates single points of failure. Data replication however induces a communication cost to maintain consistency among replicas. Eventually-Serializable Data Services (ESDS) [5] is a system that was formulated to help reduce these costs. The algorithm implementing ESDS allows the users to selectively relax the consistency requirements in exchange for improved performance. Given a definition of an arbitrary serial data type, ESDS guarantees that the replicated data will eventually be consistent (i.e., presenting a single-copy centralized view of the data to the users), and the users are able to require the results for certain operations to be consistent with the stable total order of all operations.

The design, analysis, and implementation of systems such as ESDS is not an easy task, and specification languages have been developed to express these algorithms formally. For instance, the framework of (timed) I/O automata [9,7] and their associated tools [10] allows theorem provers (e.g., PVS [13]) and model

W.-J. van Hoeve and J.N. Hooker (Eds.): CPAIOR 2009, LNCS 5547, pp. 193–207, 2009.

checkers (e.g., UPPAAL [8,3]) to reason about correctness. The ESDS algorithm is in fact formally specified with I/O automata and proved correct [5]. Once a specification is deemed correct, it must be implemented and deployed. The implementation typically consists of communicating software modules whose collective behaviors cannot deviate from the set of acceptable behaviors of the specification; see [4] for a methodic implementation of the algorithm and a study of its performance. The deployment then focuses on mapping the software modules onto a distributed computing platform to maximize performance.

This research focuses on the last step: the deployment of the implementation on a specific architecture. The deployment can be viewed as a resource allocation problem in which the objective is to minimize the network traffic while satisfying the constraints imposed by the distributed algorithms. These constraints include, in particular, the requirements that replicas cannot be allocated to the same computer since this would weaken fault tolerance. The basic ESDS Deployment Problem (ESDSDP) was considered by [2] and was modeled as a MIP with disappointing results even on small instances. A highly competitive CP approach as well as a viable MIP model can be found in [11]. In [12], the basic ESDSDP model was extended to take into account bandwidth constraints on various segments of the network interconnect. This richer model proved harder for both MIP and CP solvers with running times in hours on some instances.

This paper focuses on the bandwidth-limited version of the ESDSDP and studies a constraint programming (CP), a Constraint-Based Local Search (CBLS), and an hybrid model. The empirical evaluation demonstrates that the CP model solves most instances in minutes (significantly outperforming the earlier CP model) and that the CBLS model can deliver high-quality solutions quickly. The CP model is a natural encoding of ESDSDP together with a simple search heuristic focusing on the objective. It improves earlier results [11] by exploiting a dominance property to rule out bandwidth-limited paths that are provably inferior to already considered paths. The CBLS model uses the same natural declarative model and its search procedure uses two neighborhoods that focus on the objective and the feasibility part of the model. The feasibility neighborhood is a simple constraint-directed search. The hybrid delivers optimality proofs for the biggest instances from 200 to 2800 times faster than the MIP and 30 to 90 times faster than the earlier CP model while improving the robustness.

The rest of this paper is organized as follows. Section 2 presents an overview of the bandwidth-limited ESDS and illustrates the deployment problem on a basic instance. Section 3 introduces the high-level deployment model and Section 4 presents the CP models. Section 5 presents the CBLS model, while Section 6 covers the hybridization. Section 7 reports the experimental results and analyzes the behavior of the models in detail. Section 8 concludes the paper.

2 Deployment of Eventually-Serializable Data Services

An Eventually-Serializable Data Service (ESDS) consists of three types of components: *clients*, *front-ends*, and *replicas*. Clients issue requests for operations

on shared data and receive responses returning the results of those operations. Clients do not communicate directly with the replicas; instead they communicate with front-ends which keep track of pending requests and handle the communication with the replicas. Each replica maintains a complete copy of the shared data and "gossips" with other replica to stay informed about operations that have been received and processed by them. The clients may request operations whose results are tentative, but can be quickly obtained, or they can request "strict" operations that are possibly slower, but whose results are guaranteed to be consistent with an eventual total order on the operations. Each replica maintains a set of the requested operations and a partial ordering on these operations that tends to the eventual total order on operations. Clients may specify constraints on how the requested operations are ordered. If no constraints are specified by the clients, the operations may be reordered after a response has been returned. A request may include a list of previously requested operations that must be performed before the currently requested operation. For any sequences of requests issued by the clients, the service guarantees eventual consistency of the replicated data [5].

ESDS is well-suited for implementing applications such as a distributed directory service, such as Internet's Domain Name System [6], which needs redundancy for fault-tolerance and good response time for name lookup but does not require immediate consistency of naming updates. Indeed, the access patterns of such applications are dominated by queries, with infrequent update requests. Optimizing the deployment of an ESDS application can be challenging due to non-uniform communication costs induced by the actual network interconnect, as well as the various types of software components and their communication patterns. In addition, for fault tolerance, no more than one replica should reside on any given node. There is a tradeoff between the desire to place front-ends near the clients with whom they communicate the most and the desire to place the front-ends near replicas. Note also that the client locations may be further constrained by exogenous factors. Deployment instances typically involve a handful of front-ends to mitigate between clients and servers, a few replicas, and a few clients. Instances may not be particularly large as the (potentially numerous) actual users are *external* to the system and simply forward their demands to the *internal* clients modeled within the ESDS. A significant additional complication in deriving deployment mappings is due to bandwidth limitations placed on connections between the nodes in the target platform, e.g., a subnet based on a switched ethernet at 100Mbit/s.

Figure 1 depicts a simple ESDP Deployment Problem (ESDSDP). The left part of the figure shows the hardware architecture, which consists of 10 heavy-duty servers connected via a switch (full interconnect) and 4 "light" servers connected via direct links to the first four heavy-duty servers. For simplicity, the cost of sending a message from one machine to another is the number of network hops. For instance, a message from PC_1 to PC_2 requires 3 hops, since a server-to-server message through the switch requires one hop only. The right part of Figure 1 depicts the abstract implementation of the ESDS. The ESDS software

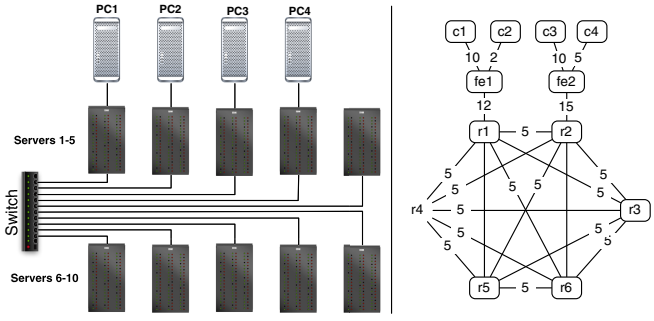

Fig. 1. A Simple ESDS Deployment Problem

modules fall in three categories: (1) client modules that issue queries $(c_1, \cdots, c_4)$; (2) front-end modules (fe_1, fe_2) that mediate between clients and servers and are responsible for tracking the sequence of pending queries; and (3) replicas $(r_1, \cdots, r_6)$. Each software module communicates with one or several modules, and the right side of the figure specifies the volume of messages that must flow between the software components in order to implement the service. The problem constraints in this problem are as follows: the first 3 client modules must be hosted on the light servers $(PC_1, \cdots, PC_4)$ while the remaining components $(c_4, fe_1, fe_2, r_1, \cdots, r_6)$ must run on the heavy-duty servers. Additionally, the replicas $r_1 \cdots r_6$ must execute on distinct servers to achieve fault tolerance. The deployment problem consists of finding an assignment of software components to servers that satisfies the constraints above and minimizes the overall network traffic expressed as the volume of messages sent given the host assignments.

3 Modeling Optimal ESDS Deployments

The deployment model for ESDS is based on [1,2]. The input data consists of:

- The set of software modules C;
- The set of hosts N;
- The subset of hosts to which a component can be assigned is denoted by booleans $s_{c,n}$ equal to *true* when component c can be assigned to host n;
- The network cost is directly derived from its topology and expressed with a matrix h where $h_{i,j}$ is the minimum number of hops required to send a message from host i to host j. Note that $h_{i,i} = 0$ (local messages are free);
- The message volumes. In the following, $f_{a,b}$ denotes the average frequency of messages sent from component a to component b;
- The separation set *Sep* which specifies that the components in each $S \in Sep$ must be hosted on a different servers;
- The co-location set *Col* which specifies that the components in each $S \in Col$ must be hosted on the same servers;

The decision variables x_c are associated with each module $c \in C$ and $x_c = n$ if component c is deployed on host n. An optimal deployment minimizes

$$\sum_{a \in C} \sum_{b \in C} f_{a,b} \cdot h_{x_a, x_b}$$

subject to the following. Components may only be assigned to supporting hosts

$$\forall c \in C \ : \ x_c \in \{i \in N \mid s_{c,i} = 1\}.$$

For each separation constraint $S \in Sep$, we impose $\forall i, j \in S \ : \ i \neq j \Rightarrow x_i \neq x_j$. Finally, for each co-location constraint $S \in Col$, we impose $\forall i, j \in S \ : \ x_i = x_j$.

Realistic target networks may impose bandwidth limitation on connections between hosts due to either physical channel limitations, input buffer limitations, or QoS guarantees. To reflect such constraints, the deployment model is extended to incorporate bandwidth limitations on some connections. Informally, a *connection* is a network inter-connect between a set of k machines. For instance, a connection can be a dedicated point-to-point link or a switched 802.11-wired Ethernet subnet. A deployment platform then reduces to a set of *connections* with some nodes (e.g., routers or machines with several network cards) appearing in several connections to establish bridges. More formally, a deployment platform is an hypergraph $H = (X, E)$ where X is the set of nodes and E is a set of hyperedges, i.e., $E \subseteq \mathcal{P}(X) \setminus \{\emptyset\}$ where $\mathcal{P}(X)$ is the power-set of X. Each hyperedge c carries a bandwidth capacity that is denoted by $c.bw$ and its vertices are denoted by $c.nSet$. If a hyperdge c (connection) has no bandwidth limitations, its bandwidth capacity is $c.bw = 0$.

Each pair of hosts is connected by one or more paths, where a path is an ordered collection of hyperedges (connections) from the source to the destination host. A path is bandwidth-limited if one or more of its hyperedges (connections) has limited (positive) bandwidth; otherwise a path is not bandwidth-limited. The network paths are represented by the following calculated parameters:

- $P_{i,j}$ denotes the set of paths from host i to host j for all $i, j \in N$. If $i = j$, $P_{i,j}$ contains a single path of zero length;
- A 3-D matrix h, where $h_{i,j,p}$ is the length of path p from host i to host j (replacing the 2-D hops matrix h);
- A boolean matrix $hasC$, where $hasC_{i,j,p,c}$ is *true* if path p from host i to host j uses the hyperedge (connection) c.

The model contains decision variables to choose the paths to be used by communicating software modules deployed on hosts. Specifically, a new decision variable $path_{a,b}$ is associated with each communicating pair of components a and b and represents the path component a uses to send messages to component b. This variable must satisfy the constraint $path_{a,b} \in P_{x_a, x_b}$. This reflects the current assumption that each pair of components uses a single directed path for data transmission. However, $path_{a,b}$ and $path_{b,a}$ may be different.

An optimal deployment minimizes

$$\sum_{a \in C} \sum_{b \in C} f_{a,b} \cdot h_{x_a, x_b, path_{a,b}}$$

subject to the supporting, separation, and co-location constraints presented earlier and the following bandwidth constraint for each $c \in E$ with $c.bw > 0$:

$$\sum_{a \in C} \sum_{b \in C} f_{a,b} \cdot hasC_{x_a, x_b, path_{a,b}, c} \leq c.bw$$

4 The CP Model

The COMET program for bandwidth-limited connections is shown in Figure 2. The data declarations are specified in lines 2–11, and the decision variables are declared in lines 12–13. Variable $x[c]$ specifies the host of component c, with its domain computed from the support matrix s. The variable $path[c1, c2]$ specifies the path used to send messages from component $c1$ to component $c2$, expressed as the rank of the selected path in the set $P[x[c1], x[c2]]$.

Lines 14–18 specify the objective function, which minimizes communication costs. The CP formulation uses a three-dimensional element constraint since the matrix h is indexed not only by variables for the two hosts but also by the variable for the particular communication path used between them.

Lines 19–24 contain the co-location and separation constraints. Lines 25–26 limit the ranges of the individual path variables to the number of paths between the hosts onto which the components are deployed. Lines 27–29 are the bandwidth constraints: for each hyperedge $c \in E$, the bandwidth $c.bw$ must be greater than or equal to the sum of the communication frequencies of all pairs of components with c in their chosen path. The `onDomains` annotations indicate that arc-consistency must be enforced for each constraint.

The search procedure, depicted in lines 31–45, operates in two phases. In the first phase (lines 31–40) all the components are assigned to hosts, beginning with the components that communicate most heavily. The search must *estimate* the communication cost between components a and b's potential deployment sites along *any* given path. Line 33 picks the first site k for component b, and the `tryall` instruction on line 34 considers the sites for component a in increasing order of path length based on an estimation equal to the shortest path one could take between the choice n and the selection k. (Symmetry breaking as in [11] optionally may be included.) The second phase (lines 41-45) labels the path variables, backtracking as needed over the initial component assignments.

Path Dominance. The initial CP bandwidth model [12] used a weak form of dominance. After finding a path that is not bandwidth-limited for a given pair of nodes, it discards all the other paths between those two nodes that are of the same length or longer. Longer paths need not be considered because the deployment model minimizes the number of "hops" for transmitted messages, and a shorter, non-bandwidth limited path always will be a better choice.

```
Solver<CP> cp();
range C = ...; // The components
range N = ...; // The host nodes
int[,] s = ...; // The supports matrix
int[,] f = ...; // The frequency matrix
int[,,] h = ...; // The hops matrix
set{set{int}} Sep = ...; // The separation sets
set{set{int}} Col = ...; // The co−location sets
set{connection} Conn = ...; // The connections
set{connection}[,] P = ...; // The paths matrix
int[,,,] hasC = ...; // The path/connection matrix
var<CP>{N} x[c in C](cp, setof(n in N) (s[c,n] == 1));
var<CP>{int} path [a in C, b in C](cp,0..max(i in N, j in N) P[i,j].getSize()−1);
var<CP>{int} obj(cp, 0..1000);
minimize<cp> obj
subject to {
   cp.post(obj == sum(a in C, b in C: f[a,b] != 0) f[a,b] ∗ h[x[a],x[b],path[a,b]],
      onDomains);
   forall(S in Col)
      select(c1 in S)
         forall (c2 in S: c1 != c2)
            cp.post(x[c1] == x[c2], onDomains);
   forall(S in Sep)
      cp.post(alldifferent(all(c in S) x[c]), onDomains);
   forall (a in C, b in C : f[a,b] != 0)
      cp.post (path[a,b] < P[x[a],x[b]].getSize(), onDomains);
   forall (c in Conn: c.bw > 0)
      cp.post (c.bw >= sum (a in C, b in C: f[a,b] != 0)
         hasC[x[a], x[b], path[a,b], c] ∗ f[a,b], onDomains);
} using {
   while (sum(k in C) x[k].bound() < C.getSize()) {
      selectMax(a in C: !x[a].bound(), b in C)(f[a,b]) {
         int k = min(k in N: x[b].memberOf(k)) k;
         tryall<cp>(n in N: x[a].memberOf(n))
               by (min (i in 0..P[n,k].getSize()−1) h[n,k,i])
            cp.post(x[a] == n);
         onFailure
            cp.post(x[a] != n);
      }
   }
   forall (a in C,b in C: f[a,b] != 0 && !path[a,b].bound())
      tryall<cp> (i in 0..P[x[a],x[b]].getSize()−1) by (h[x[a], x[b], i])
         cp.post(path[a,b] == i);
      onFailure
         cp.post(path[a,b] != i);
}
```

Fig. 2. The Bandwidth-Limited Model in Comet

Several other categories of paths need not be considered in determining an optimal deployment. In particular, let $p1$ and $p2$ be any two paths between nodes $n1$ and $n2$, and let $p1.bwSet$ and $p2.bwSet$ be the sets of bandwidth-limited connections of $p1$ and $p2$. Then $p1$ can be ignored if any of the following conditions hold:

- The path $p1$ is strictly longer than $p2$ and $p2.bwSet \subseteq p1.bwSet$.
- The path $p1$ is the same length as $p2$ and $p2.bwSet \subset p1.bwSet$.
- The path $p1$ is the same length as $p2$, $p2.bwSet = p1.bwSet$, and $p1$ is ordered after $p2$ in the set of paths between $n1$ and $n2$. (This is an arbitrary selection of one of two equivalent paths.)

The extended CP model uses these rules to eliminate all dominated paths during the initialization of the model. Alternatively, these path dominance properties can be expressed as constraints on the viable paths between components. However, this strategy is ineffective. Indeed, the set of paths P is indexed by its source and destination *hosts* whereas the `path` variable is indexed by its source and destination *components*. For any two components `c1` and `c2`, the constraint on the path variables would look like `validPath[x[c1],x[c2],path[c1,c2]]==1`. This constraint, though, is unable to fully reduce the domain of the path variable until the corresponding `x`'s are fixed. Avoiding the construction of dominated path altogether alleviates that difficulty as the constraint above can simplify to an upper-bound on the size of the domain of `path`.

5 The CBLS Model

The parameters of the CBLS model (components, nodes, frequency, hops, co-located, separated, and fixed) are identical to the CP model. Likewise, the CBLS model has two decision variables: an array $x[c]$ that specifies the node on which component c is deployed, and $path[c1, c2]$ that specifies the path used to connect components $c1$ and $c2$. The search procedure of the CBLS model uses a guided local search approach which increases the weights of the constraints that are hard to satisfy. The weight variables are created when the feasibility constraints are posted. A tabu list is represented by a simple dictionary that records which variables were changed recently (the initialization per se is not shown for brevity reasons).

The declarative part of the CBLS model is shown in Figure 3. It starts with the declaration of a weighted constraint system S for the guided local search. The feasibility constraints in lines 3–19 are essentially identical to those found in the CP model. Lines 3–6 declare the co-location constraints as mere equalities, while lines 7–8 specify the separation constraints with an alldifferent. Lines 10–13 require each path variable to range over the indices of available paths between the hosts onto which the components are deployed. Finally, lines 15–19 specify the bandwidth constraints.

The core objective function O is specified in lines 21–23 as the sum of the communication frequency between each pair of components multiplied by the

```
1  WeightedConstraintSystem<LS> S(m);
2  function var{int} mkWeight(Solver<LS> m) { var{int} x(m) := 1;return x;}
3  forall (t in Col)
4    selectMin(c1 in t.cSet)(c1)
5      forall (c2 in t.cSet : c1 != c2)
6        S.post(istrue(x[c1] == x[c2]), mkWeight(m));
7  forall (s in Sep)
8    S.post(alldifferent(all(c in s.cSet)x[c]), mkWeight(m));
9
10 ConstraintSystem<LS> S2(m);
11 forall (c1 in C, c2 in C : f[c1, c2] != 0)
12   S2.post (path[c1, c2] < numPaths[x[c1], x[c2]]);
13 S.post(S2, mkWeight(m));
14
15 ConstraintSystem<LS> S3(m);
16 forall (c in 0..Conn.getSize()−1 : Conn.atRank(c).bw > 0)
17   S3.post(Conn.atRank(c).bw >= sum(c1 in C, c2 in C : f[c1, c2] != 0)
18           (hasC[x[c1], x[c2], path[c1, c2], c] ∗ f[c1, c2]));
19 S.post(S3, mkWeight(m));
20
21 FunctionSum<LS> O(m);
22 forall (c1 in C, c2 in C : f[c1, c2] != 0)
23   O.post(f[c1, c2] ∗ h[x[c1], x[c2], path[c1, c2]]);
24
25 Function<LS> C = S + O;
26 m.close();
27 weights = all(k in S.getRange()) S.getWeight(k);
```

Fig. 3. The Constraint Systems for the CBLS Model in COMET

length of the chosen path between these components. Finally the objective function C declared in line 25 combines the feasibility constraint set S with the core objective O.

The search, illustrated in Figure 4, explores two different neighborhoods. During each iteration, one of the two neighborhoods is selected by line 2 with a fixed probability *objChance* (which defaults to 70%).

The first neighborhood (lines 3–9) focuses on the objective. Line 4 selects a non-tabu variable appearing in the objective and leading to the largest decrease to the overall objective function C. The selected variable is then assigned a value that delivers the largest decrease in the objective O, and tags the variable as tabu for the next *tLen* iterations. The second neighborhood (lines 11–17) is a standard constraint-directed search that attempts to reduce the number of violations on the constraints in S. It chooses a constraint k that causes the most violations and picks the worst variable from constraint k. Line 15 then selects the most promising value for the selected variable, and line 16 reassigns it. The best feasible solution is recorded in line 22, while line 23 updates the weight of the unsatisfied constraints in S when the algorithm hasn't progressed for

```
1  while (it < maxit) {
2    if (zo.get() <= objChance) {
3      var{int}[] ox = O.getVariables();
4      selectMax(i in ox.getRange() : tabu{ox[i].getId()} <= it)(C.decrease(ox[i])) {
5        selectMin(v in ox[i].getDomain())(O.getAssignDelta(ox[i], v)) {
6          ox[i] := v;
7          tabu{ox[i].getId()} = it + tLen;
8        }
9      }
10   } else {
11     selectMax(k in S.getRange())(S.getConstraint(k).violations()) {
12       Constraint<LS> cls = S.getConstraint(k);
13       var{int}[] cx = cls.getVariables();
14       selectMax(i in cx.getRange())(cls.violations(cx[i]))
15         selectMin(v in cx[i].getDomain())(C.getAssignDelta(cx[i],v))
16           cx[i] := v;
17     }
18   }
19   it++;
20   stableit++;
21   boolean feasible = S.violations()==0;
22   if (feasible && O.value() < bestValue) saveBest();
23   if (stableit >= 100) glsUpdate();
24   if (rounds >= 200) diversify(C);
25 }
```

Fig. 4. The Search Strategy for the CBLS Model in COMET

100 consecutive iterations. Finally, line 24 performs a diversification on all the variables appearing in the objective function C whenever 200 rounds of guided local search (weights updating) were unable to further improve the objective function. The diversification simply reassigns a fraction of the variables in C (chosen based on probability $diversifyChance$) uniformly at random, resets the weights of the guided local search, and updates the bounds on the length of the tabu list.

Co-Location Preprocessing. An alternative representation of the problem replaces each co-location constraint and its associated component variables with a single component variable representing the common location of the co-located components. This avoids a large collection of equality constraints. Simple preprocessing and postprocessing steps can then recast the solution in term of the initial formulation. This leaner formulation is beneficial for the CBLS model. In the original formulation, when a component is moved, all the co-location equality constraints are violated which induces a *bump* in the optimization value, which can make these moves less desirable. The preprocessed formulation does not suffer from this problem since all the co-located components are moved as one, allowing for aggregate moves. The experimental results confirm that this representation is beneficial.

Path Dominance. Unsurprisingly, the CBLS model also can make use of the path dominance rule during the initialization of the model to eliminate paths that are provably inferior. The experimental results consider local search models with path dominance included.

6 Hybrid CBLS-CP Model

The hybrid model is a sequential composition. The CBLS model runs for 10 seconds and then passes its best solution to the CP model to complete the optimization. A hybrid model using parallel composition also was considered, where the CBLS model runs in a separate thread and notifies the CP model each time it finds a better solution. The benefit of the parallel composition is only visible on *easy* instances where the CP model proves the optimality in less than 10 seconds (and therefore stops the search right away).

7 Experimental Results

The Benchmarks. The benchmarks fall into three categories: variants of the simple ESDS deployment problem depicted in Figure 1, variants of the `HYPER8` ESDS deployment problem shown in Figure 5, and variants of the `RING6` deployment problem shown in Figure 6. The `HYPER8` and `RING6` benchmarks are studied, not because they reflect actual network configurations, but because they are simple representations of networks with many equivalent alternative paths and networks with tightly coupled hosts. To model the capabilities of the communication infrastructure of a distributed system more realistically, all the benchmarks include, in addition to the components shown, one extra software module between each pair of replicas (components $r_1, \ldots, r_6$). These extra components are "drivers" that manage the communication channels and are required to be co-located with their sending replicas. The benchmarks are as follows:

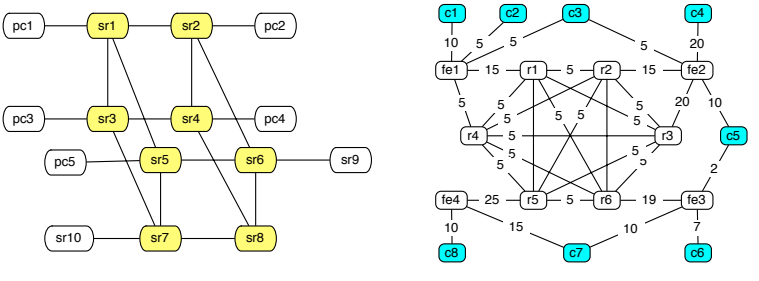

Fig. 5. Instance `HYPER8`: Deploying ESDS to a network with many equivalent paths

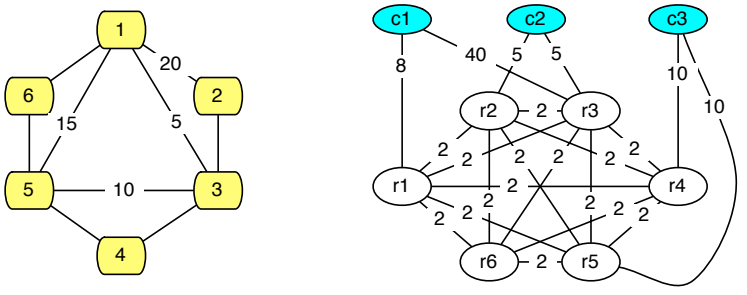

Fig. 6. Instance `RING6`: Deploying to a tightly coupled, bandwidth-limited network

- `SIM2BW1` is a variant of Figure 1 with a bandwidth limit of 5 on the connection between PC_2 and r_2.
- `SIM2BW2` is a variant of Figure 1 with a bandwidth limit of 5 on the connection between PC_2 and r_2 and a bandwidth limit of 10 on the connection between PC_3 and r_3.
- `RING4` is a variant of `RING6` with only four gossiping replicas (and no messages from $c3$ to $r5$).
- `RING5` is a variant of `RING6` with only five gossiping replicas.
- `RING6` is illustrated in Figure 6.
- `HYP8BW1` is a variant of `HYPER8` with a bandwidth limit of 10 on the connection between PC_1 and r_1.
- `HYP8BW4` is a variant of `HYPER8` with four bandwidth-limited connections.

Experimental Results for the CP Model. Table 1 reports the results for the CP model with COMET 1.1 (executing on an Intel Core 2 at 2.4Ghz with 2 gigabytes of RAM). The first three columns give the results for the initial CP bandwidth model ([12]) which only eliminates paths that are the same length or longer than the shortest non-bandwidth limited path. The next three columns give the results when the full path dominance described in Section 4 is exploited. The final three columns give the results when both path dominance and co-location

Table 1. Experimental Results for the CP Models

		Longest Path Dominance			Full Path Dominance			Path Dominance & Co-Location		
Benchmark	MIP	T_{end}	$\#Chpt$	T_{opt}	T_{end}	$\#Chpt$	T_{opt}	T_{end}	$\#Chpt$	T_{opt}
SIM2BW1	12.8	0.27	168	0.02	0.27	168	0.02	0.13	174	0.01
SIM2BW2	17.0	0.38	159	0.03	0.38	160	0.03	0.20	165	0.01
RING4	9.8	0.37	469	0.35	0.34	426	0.32	0.15	187	0.14
RING5	66.4	10.9	24422	10.4	9.6	20609	9.1	1.5	2101	1.3
RING6	327.9	132.8	300526	130.6	107.8	235161	105.6	14.4	27722	13.3
HYP8BW1	142388	4642.2	133047	893.3	123.0	43169	28.6	75.5	39137	11.3
HYP8BW4	71102	12572.9	108069	8346.5	1988.0	162227	1641.0	1175.7	103023	770.9

preprocessing are performed. Within each group, column T_{end} gives the time in seconds to find the optimum and prove optimality, column $\#Chpt$ reports the number of choice points, and column T_{opt} reports the time in seconds to find the optimum. The results are averages of 50 runs (except for the HYP8BW4 Longest Path results which are averages of 10 runs). The column MIP repeats the results (CPLEX version 11 running on an AMD Athlon at 2Ghz) for the MIP model described in [12]. It is useful to review these results in more detail.

1. Full path dominance preprocessing is faster for all benchmarks than longest path dominance preprocessing as expected. Adding co-location preprocessing also improves performance for all benchmarks.
2. For SIM2BW1 and SIM2BW2, the performance is almost identical with both path dominance techniques. There is only one path between each pair of hosts, so there are no paths to eliminate and thus no benefits.
3. RING4, RING5, and RING6 all have the same network and path characteristics. Although full path dominance only eliminates one more path than longest path dominance (20 vs. 19 eliminated paths out of 98 total paths), the impact on performance is non-negligible for all three benchmarks.
4. In HYP8BW1, full path dominance eliminates all but one path between each pair of hosts, resulting in a dramatic improvement (factor of 37) over longest path dominance which has up to 18 paths between pairs of nodes.
5. In HYP8BW4, full path dominance retains two paths between 16 pairs of hosts and one path between all other pairs. Even this relatively small number of path options make processing HYP8BW4 considerably more complex than HYP8BW1. The improvement with full path dominance is still substantial (over a factor of 6), but not as dramatic.
6. The improvement with co-location preprocessing appears to be related to the fraction of components that can be combined. The largest improvement is for RING6 where 45 components are reduced to 8, and the smallest improvement is for HYP8BW1 and HYP8BW4 where 54 components are reduced to 18.
7. With all three preprocessing techniques, the CP model finds the optimum relatively quickly for SIM2BW1, SIM2BW2, and HYP8BW1. Unfortunately, the optimum is not found until late in the search for the other benchmarks.
8. The MIP results are also likely improve with path and co-location preprocessing.

Experimental Results for the CBLS Model. Table 2 reports the results for the CBLS models. The *Opt* column gives the optimum. The $\mu(Path)$ and $\sigma(Path)$ columns report the averages and standard deviations for the best solution (Q), the time to the best solution in seconds (TB) and the total running time (TT) of the CBLS model with path dominance. The $\mu(Path\&Col)$ and $\sigma(Path\&Col)$ columns give the averages and standard deviations with both path dominance and co-location pre-processing. All the results are reported over 50 runs.

The experimental results indicate that CBLS delivers high-quality solutions in a few seconds. The elimination of the co-location constraints is beneficial in several respects. First, it reduces the running time significantly (both to termination and to the best solution), and it has a positive impact on the average

Table 2. Experimental Results for the Local Search Models

		$\mu(Path)$			$\mu(Path \,\&\, Col)$			$\sigma(Path)$			$\sigma(Path \,\&\, Col)$		
Benchmark	Opt	Q	TB	TT	Q	TB	TT	Q	TB	TT	Q	TB	TT
RING4	54	54.1	1.79	6.06	54.0	0.17	4.22	0.7	1.23	0.04	0.0	0.12	0.04
RING5	88	90.1	3.74	8.83	88.0	0.61	5.20	2.7	2.61	0.06	0.0	0.48	0.05
RING6	120	131.1	6.87	14.70	120.8	3.76	7.72	21.7	5.08	0.78	1.0	0.00	0.09
HYP8BW1	522	594.0	2.62	5.22	523.1	1.28	3.56	37.4	1.76	0.20	1.8	0.59	0.07
HYP8BW4	526	552.3	7.42	17.58	554.5	5.41	8.92	18.5	5.60	0.50	23.1	2.13	0.34

Table 3. Experimental Results for the Sequential Hybrid Models

	μ(Path)		μ(Path & Col)		σ(Path)		σ(Path & Col)	
Benchmark	T_{end}	$\#Chpt$	T_{end}	$\#Chpt$	T_{end}	$\#Chpt$	T_{end}	$\#Chpt$
RING4	10.09	11	10.03	6	0.00	1	0.01	3
RING5	11.02	88	10.14	26	0.08	15	0.01	2
RING6	23.64	17718	10.51	81	25.26	50147	0.04	7
HYP8BW1	140.70	40799	51.06	29602	9.35	4404	0.74	463
HYP8BW4	1325.02	63343	339.68	32045	294.22	28785	26.54	3675

best solution found. Indeed, as the standard deviation shows, the local search algorithm could deliver the best solution on all 50 runs on `RING4` and `RING5` and the average best solution across the board. Second, all the standard deviations improved significantly, indicating that the algorithm is more robust.

Experimental Results for the Hybrid Model. Table 3 reports the results for a sequential hybrid model that runs the best CBLS model for 10 seconds before initiating a CP search with an upper bound based on the best solution delivered in the first phase. All the results are averages based on 50 runs. The column groups are the same as for the CP model. Within each group, T_{end} denotes the time in seconds to find the optimum and prove optimality, and $\#Chpt$ denotes the number of choice points.

The first hybrid algorithm composes models relying only on the path dominance. Nonetheless, the benefit is already visible when the pure CP model is compared to the hybrid. The second hybrid composes models using both path dominance and co-location pre-processing and clearly dominates the earlier CP models. On the hardest instance (HYP8BW4) it proves optimality in 340s when the pure CP model needed 1176s. The good results can be attributed to an excellent phase 1 that delivers a high quality solution to bootstrap the second phase. On the ring instances the runtime is dominated by the fixed 10s of local search, while on the HYPER8 it spends most of the computation in the CP phase.

8 Conclusion

This paper revisited the Bandwidth-Limited ESDS Deployment Problem and considered an improved CP model that leverages dominance properties, a CBLS model featuring the same declarative model, as well as a CP/CBLS hybrid model. The CBLS model is particularly compelling given the similarity of its declarative

part and its ability to deliver high-quality solutions quickly. Its search procedure composes a standard constraint-directed neighborhood for the feasibility part of the model with a tabu-based greedy gradient descent for the objective function. The path dominance and the co-location preprocessing steps proved very effective for CP and CBLS, both in terms of the solution quality and the time to solve the model. Constraint Programming now appears to be the ideal methodology to solve this class of problem for which hard instances can be solved to optimality in 5 to 10 minutes.

Acknowledgements. This work was partially supported through the following NSF awards: DMI-0600384, IIS-0642906 and CCF-0702670 as well as an ONR award N000140610607 and support from AFOSR under contract FA955007C0114.

References

1. Bastarrica, M., Demurjian, S., Shvartsman, A.: Software architectural specification for optimal object distribution. In: SCCC 1998: Proc. of the XVIII Int-l Conf. of the Chilean Computer Science Society, Washington, DC, USA (1998)
2. Bastarrica, M.C.: Architectural specification and optimal deployment of distributed systems. PhD thesis, University of Connecticut (2000)
3. Behrmann, G., David, A., Larsen, K., Möller, O., Pettersson, P., Yi, W.: UPPAAL - present and future. In: Proceedings of the 40th IEEE Conference on Decision and Control (CDC 2001), pp. 2881–2886 (2001)
4. Cheiner, O., Shvartsman, A.: Implementing an eventually-serializable data service as a distributed system building block. Networks in Distributed Computing 45, 43–71 (1999)
5. Fekete, A., Gupta, D., Luchangco, V., Lynch, N.A., Shvartsman, A.A.: Eventually-serializable data services. Theor. Comput. Sci. 220(1), 113–156 (1999)
6. IETF. Domain name system, rfc 1034 and rfc 1035 (1990)
7. Kaynar, D.K., Lynch, N., Segala, R., Vaandrager, F.: The Theory of Timed I/O Automata. Synthesis Lectures in Computer Science. Morgan & Claypool Publishers (2006)
8. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. International Journal on Software Tools for Technology Transfer 1(1-2), 134–152 (1997)
9. Lynch, N., Tuttle, M.: An introduction to Input/Output Automata. CWI-Quarterly 2(3), 219–246 (1989)
10. Lynch, N.A., Garland, S., Kaynar, D., Michel, L., Shvartsman, A.: The Tempo Language User Guide and Reference Manual. VeroModo Inc. (December 2007), `http://www.veromodo.com`
11. Michel, L., Shvartsman, A.A., Sonderegger, E.L., Hentenryck, P.V.: Optimal deployment of eventually-serializable data services. In: Perron, L., Trick, M.A. (eds.) CPAIOR 2008. LNCS, vol. 5015, pp. 188–202. Springer, Heidelberg (2008)
12. Michel, L., Shvartsman, A.A., Sonderegger, E.L., Hentenryck, P.V.: Optimal deployment of eventually-serializable data services. CPAIOR (2008); extended version of the CPAIOR 2008 paper (submitted)
13. Owre, S., Rajan, S., Rushby, J.M., Shankar, N., Srivas, M.K.: PVS: Combining specification, proof checking, and model checking. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 411–414. Springer, Heidelberg (1996)