2.7 Distributed Protocol

In this section we describe a protocol to manage linear arithmetic constraints over a distributed system. In fact, the protocol manages resource distributions as a mechanism to manage updates and constraint decompositions (as is described in Subsection 2.5.3). The protocol is general, in the sense that it can be applied to different network and system architectures.

First, we explain a distributed transaction primitive called RESOURCE - TRANSFER that is used in the protocol. RESOURCE-TRANSFER(i,j,$\vec{rc}$) works on a pair of sites, and transfers the *resource-contribution* $\vec{rc}$ from the *giving* site i to the *receiving* site j. The effect of the transfers is that the upper resource bound $\vec{ur}_i$ at site i will be decreased by $\vec{rc}$, after which $\vec{ur}_j$ at site j will be increased by $\vec{rc}$. It is assumed that standard distributed transaction techniques are used to assure that (1) under no circumstances (possibly involving failures) resource-contribution is added to $\vec{ur}_j$ of the receiving site before it has been reduced from $\vec{ur}_i$ of the giving site, and (2) the standard ACID properties (i.e., atomicity, consistency, isolation, and durability) of distributed transactions. Important to note is that distributed transaction protocols to ensure these properties (e.g., two phase commit) are less expensive for transactions involving two sites only, as is done in our RESOURCE-TRANSFER.

We can now provide the basic assumptions for our protocol which are as follows:

Distributed Protocol Assumptions

1. The global database is abstracted by real values for the vector of variables $\vec{x} = (\vec{y}_1, \ldots, \vec{y}_M)$, where $(\vec{y}_1, \ldots, \vec{y}_M)$ is a partition of $\vec{x}$ [12].

2. A set of M distributed sites, where at each site i, $1 \leq i \leq M$, variables $\vec{y}_i$ are maintained. Since $(\vec{y}_1, \ldots, \vec{y}_M)$ is a partition of $\vec{x}$, i.e., there is no variable replication [13], i.e., $\vec{y}_i \cap \vec{y}_j = \emptyset$, for every $i \neq j$. Furthermore, each site i, $1 \leq i \leq M$, has a local transaction manager that guarantees the standard ACID properties (i.e., atomicity, consistency, isolation, and durability), as well as back-up and recovery from failure.

3. The global constraint is of the form $\Omega = A\vec{x} \leq \vec{b}$, i.e., Ω is a system of linear constraints. Local constraints are given by compact splits of Ω. Every site i maintains in addition to its local instance $\vec{y}_i^0$ and the global constraint Ω, the triple $(\vec{lr}_i, \vec{r}_i, \vec{ur}_i)$, where $((\vec{lr}_1, \vec{r}_1, \vec{ur}_1), \ldots, (\vec{lr}_M, \vec{r}_M, \vec{ur}_M))$ is a permissible resource distribution (thus the local constraint $A_i\vec{y}_i \leq \vec{r}_i$ is also implicitly given). Safe (re-) decompositions will be done by updating the permissible resource distribution.

4. When an update is required at site k, and if the new update (i.e., for $\vec{y}_i$) satisfies the current local constraint at site k, the update is performed. If it does not satisfy its local constraint, then, site k designates one site as a *coordinator* and

[12] Any database model, e.g., relational or object-oriented, can be used, we assume that values for $\vec{x}$ are either explicitly stored in the actual database or expressed as views (e.g., aggregations).
[13] If there is replication, i.e., a variable x appears at two sites, it can be reduced to a non-replica case by replacing x with x_1 and x_2 and adding the constraint $x_1 = x_2$ to Ω.

sends to it a request for the minimal resource-contribution necessary to make the update (i.e., by Proposition 9, the minimal $\vec{rc}_k$ is $\vec{ur}_k - \vec{lr}'_k$, where $\vec{lr}'_k$ is the new lower bound of k reflecting the new update).

5. The task of a coordinator is to facilitate a local update at site k (when the local update does not satisfy the local constraint C_k), coordinating the (re-) decomposition process. This is done by finding a set of sites θ, containing site k, and trying to create a new permissible θ-resource distribution, as follows. The coordinator asks for resource contribution from sites until either:

   (a) A subset of connected [14] (to the coordinator) and operational sites θ is found, for which $\vec{lr}'_\theta \leq \vec{ur}_\theta$, where $\vec{lr}'_\theta$ is the cumulative lower bound for θ (i.e., $\vec{lr}'_\theta = \sum_{i\in\theta} \vec{lr}'_i$) if the new update(s) in θ were reflected, (i.e., by Proposition 9, there exists a compact split of Ω satisfying resource partition $\vec{ur}_\theta$ and local consistency w.r.t. database instances in θ), and a new permissible θ-resource distribution is created, or

   (b) For the maximal set θ^* of sites that are connected (to the coordinator) and operational, and $\vec{lr}'_{\theta^*} \not\leq \vec{ur}_{\theta^*}$ (i.e., by Proposition 9 there does not exist a compact split of Ω satisfying resource partition $\vec{ur}_\theta$ and local consistency w.r.t. database instances in θ). Then, the update is refused.

In case (a), resources will be re-distributed in the optimal way in the sense of

[14] By "connected" we mean that every site on θ can send messages to any other site in θ.

Theorem 5.

6. Exchanging resource contributions (thus modifying the current upper resource bounds in the resource distribution) is only done via the distributed transaction primitive RESOURCE-TRANSFER. Each *giving* site can provide a resource contribution $\vec{rc}$ such that $\vec{0} \leq \vec{rc} \leq \vec{ur}_i - \vec{lr}_i$, i.e., local consistency will still be preserved. When the upper resource bound $\vec{ur}_i$ is reduced or increased at site i, the local protocol adjusts its resource $\vec{r}_i$ accordingly (see Section 2.5.3).

7. Failure model: both sites and communication links may fail, but persistent storage does not fail. We assume that the site failures stop site execution without performing any incorrect actions. Communication link failures may separate the sites into more than one connected component of communicating sites (θ's).

First, we discuss the properties we would like to guarantee, and then, a single coordinator protocol is presented as a specific architecture case implementation.

2.7.1 Properties

When a local constraint is violated, our protocol performs distribute processing. In order to guarantee a correct distributed processing, we propose the following properties.

CSOL Properties

1. Global and Local Consistency: every database instance $\vec{x} = (\vec{y}_1, \ldots, \vec{y}_M)$ must satisfy the global constraint Ω; every local instance $\vec{y}_i$ at every site i must satisfy the local constraint C_i.

2. Partial (θ-)Decomposition Soundness: If the protocol performs a (re-) decomposition of constraints in θ, then there must exist a safe (compact split) decomposition of Ω that satisfies resource partition $\vec{ur}_\theta$ (i.e., the global resource bound in θ) and local consistency w.r.t. the current database instance.

3. Last-Resort Update Refusal: Let $\vec{y}_i^0$ be a new update for site i, and let θ^* be a maximal set of operational and connected sites that contains the site i (i.e., no resources outside θ^* are available). Last-Resort Update Refusal means that the protocol refuses the update $\vec{y}_i^0$ at site i only if there does not exist a safe (compact split) decomposition of Ω that satisfies resource partition $\vec{ur}_{\theta^*}$ (i.e., the global resource bound in θ^*) and local consistency w.r.t. the current database instance.

4. θ-Optimality: if a protocol performs (re-)decompositions of sites in θ, it is must be optimal in the sense of Theorem 5.

Global and local consistency are standard properties that we would like to preserve. Partial θ-Decomposition Soundness says, intuitively, that the protocol does the best under the circumstances, i.e., using only the knowledge of resources at sites in θ. It also

implies that resources are not lost because of failures, because Partial θ-Decomposition Soundness must hold at all times, including times after (local) recoveries from failures. Finally, Last-Resort Update Refusal says that the protocol refuses updates (and re-decompositions), only when there is no choice under the circumstances, i.e., no site outside of θ^* can be reached (that is, a θ^* is maximal set) and, based just on the information at sites in θ^*, we cannot guarantee satisfaction of global and local consistency.

Now we are able to present the main result of this section, which is a theorem guaranteeing CSOL-properties under the distributed protocol assumptions.

Theorem 6. *Any protocol that uses exclusively $RESOURCE-TRANSFER$ primitive for exchanging resources, is guaranteed to satisfy (1) safety and local consistency (and thus global consistency), (2) partial θ-decomposition soundness, (3) last-resort update refusal, and (4) θ-decomposition optimality.*

Proof. Local consistency follows from the fact that RESOURCE-TRANSFER can only reduce resources such that local consistency is satisfied (Distributed Protocol Assumption 6). Global consistency follows from the fact that RESOURCE-TRANSFER never create resources (i.e., $\vec{rc} \leq \vec{ur}_i - \vec{lr}_i$) and it first reduces resource-contribution from the giving site and then add it to the receiving site (RESOURCE-TRANSFER assumption). Therefore, the protocol only produces permissible resource distribution. Finally, partial θ-decomposition soundness, last-resort update refusal, and θ decomposition optimality follow directly from assumption 5) of our Distributed Protocol Assumptions. □

We exemplify an instance of the protocol, which has a single coordinator, in the next subsection.

2.7.2 Protocol with one Coordinator

Here we exemplify an instance of the distributed protocol for one-coordinator architecture. The suggested protocol is based on RESOURCE-TRANSFER primitive, and the Distributed Protocol Assumptions. In addition to that, we assume the following.

1. Coordinator corresponds to site p. Since, there is only one coordinator, it knows the complete resource distribution among sites, and time to time decides to (1) collect information from sites (i.e., lower bounds), and (2) re-decompose without any non-coordinator site requirement.

2. Each site i has an underlying layer mechanism to inform whether or not a site j is connected and operative. We assume that such a mechanism triggers a variable $ALERT_{ij}$ which indicates that site j is disconnected (w.r.t. site i) or inoperative.

Now we are ready to describe the implementation of our protocol at non coordinator (regular) and coordinator sites.

Activities by Non-Coordinator Sites

Each site checks if local updates satisfies the current local constraints. When an update $\vec{y}_k$ arrives to site k, this site performs the following.

1. (a) If $\vec{y}_k$ satisfies the local constraint, then perform the update.

   (b) If $\vec{y}_k$ does not satisfy the local constraint. Then, site k sends a request to the coordinator with the necessary resource-contribution $\vec{rc}_k$ to make the update. This resource-contribution is calculated as follows:

$$\vec{rc}_k = \begin{cases} \vec{A}_{ki}\vec{y}_k - ur_{ki} & \text{if } \vec{A}_{ki}\vec{y}_k > ur_{ki}, \\ 0 & \text{otherwise.} \end{cases}$$

   Then, site k waits until the coordinator provides the resource-contribution needed or $ALERT_{kp}$ is triggered.

2. When site k receives resource-contribution from the coordinator, then the resources are updated. If there are enough resources, then the update is performed. Otherwise, site k rejects the update.

3. If site k receives the $ALERT_{kp}$ saying that the coordinator is not connected or operative, site k reject the update.

When a site k receives a request from the coordinator, asking for resource - contribution, site k performs the following:

1. Site k decides resource-contribution $\vec{rc}_k$ to provide to the coordinator as follows: $\vec{0} \leq \vec{rc}_k \leq max\{\vec{0}, \vec{ur}_k - \vec{lr}_k\}$. Then, site k initiates RESOURCE-TRANSFER(k,p,$\vec{rc}_k$) to the coordinator.

Activities by Coordinator p

Coordinator p maintains set ξ for all sites requesting resource-contribution from it, and a vector $\vec{rc}_p$ with the current resource-contribution holds by the coordinator. Then, the coordinator performs the following:

1. When a request $\vec{rc}_k$ is received from a site k, and the coordinator has enough resources (i.e., $\vec{rc}_p \geq \vec{rc}_k$) and ξ is empty. Then, the coordinator reduces $\vec{rc}_k$ from $\vec{rc}_p$ and provides resource-contribution $\vec{rc}_k$ to site k using primitive RESOURCE-TRANSFER(p,k,$\vec{rc}_k$). Otherwise, the following is performed.

2. The coordinator include site k in ξ and decides an initial set θ of sites to ask for resource-contribution. For each site $i \in \theta$ sends a request asking for it. Coordinator waits until the resource-contributions arrive or $ALERT_{pi}$ is triggered.

3. All possible answers from θ have been received (i.e., all connected and operational sites in θ have ended RESOURCE-TRANSFER primitive), and there are no enough resources, i.e., $\vec{lr}'_\theta \not\leq \vec{ur}_\theta$. Then,

   (a) θ can be increased, the coordinator increases it and sends a request asking for resource-contribution for the new sites in θ.

   (b) θ can not be increased. Then, for each site $i \in \xi$, the coordinator initiates primitive RESOURCE-TRANSFER(p,i,$\vec{0}$), and deletes site i from ξ.

4. If enough resources have been collected, i.e., i.e., $\vec{lr}'_\theta \leq \vec{ur}_\theta$, coordinator decides

according to Theorem 5, how much resource-contribution $\vec{rc}_i^*$ is distributed to sites in ξ and θ. Then, for each site initiates RESOURCE-TRANSFER(p,i,$\vec{rc}_i^*$) and deletes sites from ξ.

This implementation satisfies the Distributed Protocol Assumptions (see Section 2.7) and uses the primitive RESOURCE-TRANSFER to exchange resources, thus by Theorem 6 it satisfies all our CSOL-properties. The following section presents the implementation and some experiment of our optimization framework.

2.8 Algorithms, Implementation and Experiments

This section presents a general algorithm to solve the optimization problems presented in Sections 2.4 and 2.5. Experimental results are presented for a single partition case. We use a set of experimental linear systems to show the algorithm behavior for varying numbers of constraints and variables. We use the problem size (product of the number of variables and the number of constraints) and the algorithm's running time as main measures. The algorithm was implemented using visual C++ 4.0, and was run it on a 120 Mhz PC compatible.

The optimization problem has linear constraints, and a non-linear objective function, based on volume representation. In general, volume representation is based on vertex enumeration (implicit or explicit), or recursive representations [Las83, Bea96], where some of its properties are: positive homogeneous function of its right-hand-side

vector, local convexity, and local concavity. However, these properties are not enough to guarantee optimal solutions using a global search algorithm.

We use a local search algorithm [LH96, Glo89, Jea91] to solve our optimization problem. The structure is as follows: a number of local searches are performed, where for each one, the algorithm checks if the local optimum is better than the current objective function value. This procedure is repeated until there is no acceptable neighborhood possible, i.e., a neighborhood where the objective function is locally concave.

Minimally-Constrained Safe Decomposition Algorithm

Let Pr be a subset of properties $\{compactness, lc, pcp, rp\}$, where properties *compactness* or *rp* must be included, and $\theta = \{k+1, \ldots, M\}$ be a subset of sites $\{1, \ldots, M\}$.

- Step 0. Assign an initial solution to $\vec{r}_{k+1}, \ldots, \vec{r}_M$, the objective function $f^*_{Pr} = f_t(D(\vec{r}_{k+1}, \ldots, \vec{r}_M))$, and select an initial acceptable neighborhood s.

- Step 1. Perform a local search in s, obtain the solution $\vec{h}^*_{k+1}, \ldots, \vec{h}^*_M$ of

$$maximize f_{Pr}(D(\vec{h}_{k+1}, \ldots, \vec{h}_M))$$

$$s.t.\ \Phi_{Pr}(\vec{h}_{k+1}, \ldots, \vec{h}_M) \wedge$$

$$(\vec{h}_{k+1}, \ldots, \vec{h}_M) \in s$$

- Step 2. If $f_{Pr}(D(\vec{h}^*_{k+1}, \ldots, \vec{h}^*_M)) > f^*_{Pr}$, then $f^*_{Pr} = f_{Pr}(D(\vec{h}^*_{k+1}, \ldots, \vec{h}^*_M))$, and $\vec{r}_i = \vec{h}^*_i, (k+1) \leq i \leq M$.

- Step 3. Select a new acceptable neighborhood s, and go to step 2. If there is no acceptable neighborhood, go to step 4.

- Step 4. Report the solution as: objective function f^*_{Pr}, and the solution $\vec{r}_{k+1}, \ldots, \vec{r}_M$.

The local search (step 1) is a non-linear optimization problem, with concave objective function. We use the Frank-Wolfe algorithm [BS79, Kam84], and volume algorithms [Las83, Bea96] to calculate at each iteration the gradient of f_{Pr}. This algorithm is as follows:

- Step 1.1 Let $q \leftarrow 0$ be an iteration index, and ϵ a stop condition. Obtain an initial solution and assign it to $\vec{w}_0$,

- Step 1.2 Obtain the solution of the following linear optimization problem:

$$Max\ \nabla f_{Pr}(D(\vec{w}_q))\ (\vec{h}_{k+1}, \ldots, \vec{h}_M)$$

$$s.t.\ \Phi_{Pr}(\vec{h}_{k+1}, \ldots, \vec{h}_M)$$

  where $\nabla f_{Pr}(D(\vec{w}_q))$ is the gradient of f_{Pr} evaluated at the point $\vec{w}_q$. Analyze the solution, and if: (a) it is not feasible, then the original system is infeasible, then stop, and (b) there exists the solution, then assign it to $\vec{v}$.

- Step 1.3 Get the optimal step λ^* as the solution of the problem:

$$Max\ f_{Pr}(D(\lambda\vec{v} + (1-\lambda)\vec{w}_q))$$

$$s.t.\ 0 \leq \lambda \leq 1$$

- Step 1.4 Assign to $w_{q+1} \leftarrow \lambda^*\vec{v} + (1-\lambda^*)\vec{w}_q$, if $\parallel w_{q+1} - w_q \parallel \leq \epsilon$ then go to step 1.5, or $q \leftarrow q+1$, and go to step 1.2.

- Step 1.5 Report $\vec{w}_{q+1}$ as the solution.

Note that steps 1.2 and 1.3 require additional algorithms. We use the simplex algorithm [BS79, CZ96] to solve the linear problem (step 1.2), and we use the bisecting search method [BS79] to solve the one-variable optimization problem (step 1.3).

The algorithm's complexity depends on the complexity of finding an acceptable neighborhood, and for each neighborhood visited, the complexity of n linear programs (Frank-Wolfe algorithm), and for each linear program, the complexity of M volume calculations. However, volume calculation complexity is mitigated for the smaller matrix range associated with each partition element.

Note that for the schema-based and individual partition case, we have only one acceptable neighborhood, and therefore we need to solve just step 2. The results for this special case are depicted in Table 2.1.

Table 2.1 summarizes the results for 21 experiments, where P is the problem number assigned, N is the number of variables, M the number of constraints, and Time is the

Table 2.1: Empirical Results

P	N	M	Time	P	N	M	Time
1	4	3	0.66	12	50	50	9.01
2	7	7	0.30	13	80	80	40.86
3	12	7	0.66	14	100	100	92.16
4	100	10	1.05	15	6	5	0.44
5	8	6	12.00	16	12	7	1.10
6	18	6	1.21	17	21	10	1.76
7	8	5	1.87	18	33	14	3.79
8	6	6	0.39	19	55	16	10.21
9	10	10	0.88	20	80	18	20.38
10	20	20	0.94	21	100	20	35.05
11	30	30	2.58	-	-	-	-

running time measured in seconds. The general structure of the experiments is as follows: experiments from 1 to 7 are random linear systems (with at least one solution), experiments from 8 to 14 correspond to scheduling problems, and experiments from 15 to 21 have a transportation problem structure.

Figure 2.3 shows that the running time does not behave exponentially in the size of the problem (notice that the x axis is the natural logarithm of the problem size). However, there is a high variance in this relation. The following figures present the time for each class of problem.

However, the relation between problem size and run time for the transportation type problem is clearer as shown in Figure 2.4. One observes a smoother relation between these variables.

Figure 2.5 shows the relation between these variables for the scheduling type problem. In the same way that the transportation case, this relation is smother than the

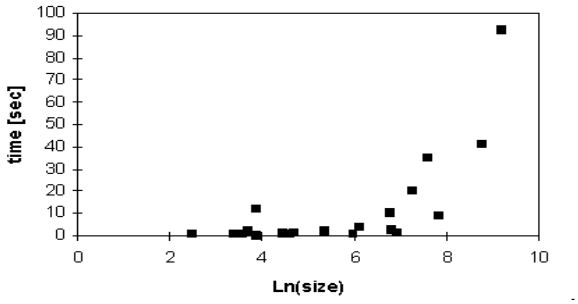

Figure 2.3: Experimental Run Time

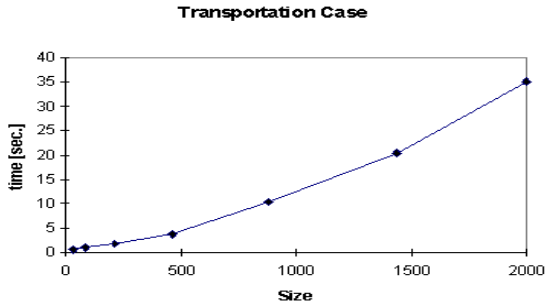

Figure 2.4: Run Time for Transportation Case

general case.

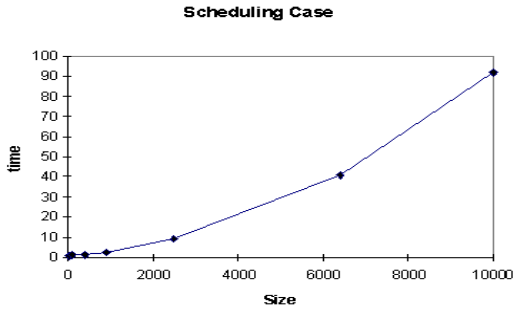

Figure 2.5: Run Time for Scheduling Case

The main conclusion of this experiment, is that the decomposition algorithm is feasible and scalable, especially when one considers that the restructuring of local constraints occurs infrequently. However, more experiments are necessary.