Computing and Informatics, Vol. 31, 2012, 1101–1125

AGENTS FOR INTEGRATING DISTRIBUTED DATA FOR FUNCTION COMPUTATIONS

Ahmed M. Khedr

Computer Sciences Department, Faculty of Science
Sharjah University, Sharjah, UAE
e-mail: akhedr@sharjah.ac.ae

Rania Mahmoud

Mathematics Department, Zagazig University
Zagazig, Egypt
e-mail: rania_mahmoud_as@yahoo.com

Communicated by Ali Akramizadeh

Abstract. Many practical problems occur when we wish to manipulate the data in a way that requires information not included explicitly in this data, and where we have to deal with functions of such a nature. In a networked environment, the data may reside in components on a number of geographically distributed sites. These databases cannot be moved to other network sites due to security, size, and privacy consideration. In this paper, we present two self-decomposing algorithms for constructing a function from given discrete data, and finding the *extrema* of any function whose arguments are stored across a number of distributed databases.

Keywords: Agents, decomposable algorithm, function optimization, horizontal fragmentation, interpolation, vertical fragmentation

Mathematics Subject Classification 2010: 68U99

1 INTRODUCTION

Computing situations that are beginning to emerge in the networked environment require data and knowledge from a number of geographically distributed sites to be considered simultaneously. A number of geographically distributed databases together form an implicitly specified global dataset that contains all the data relevant for a computation. For example, some computation tasks may require simultaneous consideration of data, parts of which reside in census databases, labor statistics databases, and employment related databases. Each of these is a huge database and resides on a different site in a different city. One cannot hope to easily move all these databases to a single computer site, merge or join them, and then execute an algorithm with the tuples in the resulting humongous database. It would be desirable to have algorithms that let the individual databases reside at their own sites and work with an imagined implicit join of the databases by decomposing themselves into localized computations such that each localized computation can be performed locally within a single site using its physical database. A common constraint in these situations is that the data cannot be moved to other network sites due to security, size, privacy and data ownership considerations.

Functions are the major tool for describing the real world in mathematical terms, and the main building block of our work. The problem of constructing a continuously defined function from given discrete data is unavoidable whenever one wishes to manipulate the data in a way that requires information not included explicitly in the data. In this age of ever increasing digitization in the storage, processing, analysis, and communication of information, it is not difficult to find examples of applications where this problem occurs (e.g. sensor network, image processing etc.). The relatively easiest and in many applications often most desired approach to solve this problem is *interpolation*, where an approximating function is constructed in such a way as to agree perfectly with the usually unknown original function at the given measurement points. Also, many practical problems occur in science, engineering, through production processes, and in the different phases of the practical life where we have a single function that depends on several independent variables, and we need to optimize this function, i.e., finding the *extrema*.

In this paper, we present a methodology for designing two decomposable algorithms for constructing a function from given discrete data, and finding the *extrema* of any function whose arguments are stored across multiple private distributed databases. This methodology consists of a general model for decomposition, and a set of algorithms for realizing this model. Our presented algorithms have the capability to decompose their computations to fit the nature of data distribution across the network sites. The objective of the algorithms is to perform function computation tasks for any arbitrary data distribution across the network by exchanging summaries derived from local databases. Our idea is to decompose the steps of an algorithm into localized computational steps that can be executed at the participating sites and the intermediate results thus obtained are transmitted

to the Coordinator site. This may have to be repeated a number of times until an algorithm is completed. It thus works by exchanging partial results during the execution of an algorithm. Our algorithms are tailored for situations in which we do not have closely connected processors. There are multiple processors but they are independent and reside at geographically distant sites. In our methodology, each data source (a network node) is represented by an agent. This agent knows all about its underlying database and can access any part of it, as represented in Figure 1, which can be described as follows:

1. Some network node Init-Node wants to perform a computation C which requires a body of data D. The entire data D may not be available on Init-Node itself.

2. A search is performed over the network to identify those other nodes that can provide some relevant parts of D for the computation and are willing to cooperate. Init-Node selects a sufficient set of participant nodes that together constitute the body D. Attributes of databases at different nodes may be unique to their sites or may exist at more than one participating sites.

3. The initiator Init-Node determines a decomposition of computations C, compatible with the distribution of attributes of D. It then seeks results of local computations from the participating nodes and composes them to construct the global result. A number of (decomposition, partial computation, composition) iterations may be required for completion of C.

If the computation does not require updates to databases then the agent also does not need an update privilege for its underlying data. The desired global computation is conveyed to the agents of the participating sites. Each agent then determines the local computations that it needs to perform keeping in mind the constraints of shared data with other sites and also the local results that it needs to share with other agents in order for the global result to evolve at Init-Node. An alternative to communicating with agents at other sites is that a single agent visits each of the participating sites and performs some local computation at each site when visited. Objectives of the agent's design include minimization of communication across the sites and enough generality of the formulation to permit agents to handle different sets of participating sites and different patterns of knowledge sharing across the participating nodes.

The rest of the paper is organized as follows: In the next section, we present the related work of our proposed problem. In Section 3, we describe the integration of the distributed data. The description of the first algorithm for constructing a function from distributed databases, its simulation, and its complexity computing are given in Section 4. Section 5 describes the second algorithm for finding the extrema of any function, its simulation results, and its complexity computing. Finally, Section 6 provides our conclusion.

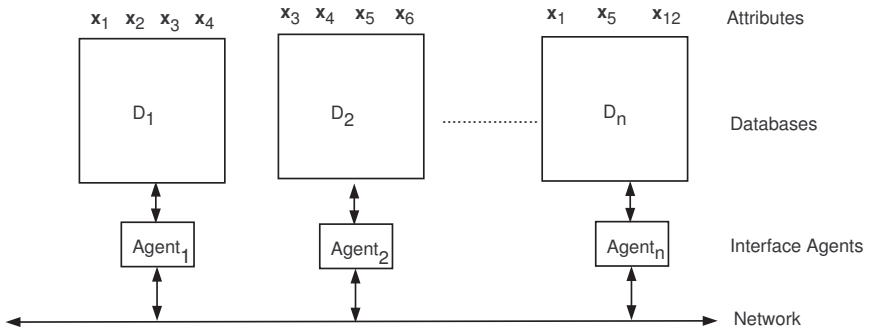

Fig. 1. Databases represented by agents

2 RELEVANT RESEARCH

There has been extensive research in algorithms for sequential and parallel architectures [1, 2, 3, 4, 5, 6, 7, 8, 9]. The main focus of the parallel and distributed algorithms has been on systems of closely coupled processors, where data can be easily shared by the processors. The distributed knowledge environment, where data cannot be shared as easily as a distributed shared memory (DSM) and must be transmitted over a wide-area network in the form of small message packets, needs a set of network algorithms, that minimize the traffic over the wide-area network. Though there are some differences between our work and DSM, the two most important differences are:

1. the optimization problem in DSM is to minimize the number of participating processors, in our work the number of processors are fixed and we looking for minimizing the number of exchanged messages among the participating nodes.
2. The processor message in DSM is only to read data, our message can do some calculation and then reply with result.

Our algorithmic decompositions can be seen as regular distributed databases algorithms being implemented by a number of coordinated agents either exchanging messages among themselves or visiting participating sites to gather results of local queries and computations [10, 11, 12, 13, 14, 15].

Multi-agent systems research has addressed many issues relating to the distribution of knowledge and processing capability over a loosely connected communication network. In most of this work [16, 17, 18] agents are modeled as having only a limited view of the global resources and knowledge. In contrast, our approach is directed at systems where cooperative agents freely access local results from other agents to evolve concepts from their collective knowledge while trying to minimize the communication of messages and data among themselves. In our algorithms, the computing agent at each site does not need to have any uncertainty about the state of data

or knowledge of other sites or computing agents. They only need to be asked for their local results and they would be truthfully given the needed information sought by other agents. However, this access can be restricted to prohibit any actual data tuples flowing out of a site. The goal of the agents in our formulation is also to minimize the exchange of information among themselves for performing the global computations.

The study of interpolation received a lot of attention since the ancient ages. In [19], Toomer believes that Hipparchus of Rhodes (190–120 BC) used linear interpolation in the construction of tables of the so-called "chord function" for the purpose of computing the positions of celestial bodies. In [20], Grevera and Udupa compared the interpolation methods for the very specific task of doubling the number of slices of 3-D medical data sets. In [21], Evans et al. presented a real interpolation method involving broken-logarithmic functors. They obtained a variety of interpolation theorems for quasi-linear operators on quasi-Banach spaces, including limiting cases. In [22], Sol and Salembier stated a quadratic image interpolation method. The formulation is connected to the optimization of lifting steps. This relation triggers the exploration of several interpolation possibilities within the same context, which uses the theory of convex optimization to minimize quadratic functions with linear constraints. In [23], James et al. presented sequential linear interpolation (SLI) structure for the efficient interpolation of multidimensional nonlinear functions. In [24], Chandrasekhar described a parallel algorithm for linear interpolation of variables in the interior pixels of a triangle when the values of the variables are specified at the vertices of the triangle. In [25], Yibin et al. considered a 3-D interpolation problem that commonly arises during structure determination for spherical viruses. They described more sophisticated global interpolation algorithms based on a least-squares point of view that exploits the symmetry. In this paper, using the sum of the squares interpolation, we present a self-decomposing algorithm for constructing a function from given discrete data of any arguments that are stored across a number of distributed databases.

In [26], Cindy, Michel and Sebastien derived expressions for the *discrete s-convex* extrema in moment spaces. The 3-convex extrema $X_{\min}^{(3)}$ and $X_{\max}^{(3)}$ have been derived and proved using Cut-criterion method that allows comparison between two random variables in the s-convex, and using the same method, the 4-convex extrema $X_{\min}^{(4)}$ and $X_{\max}^{(4)}$ have been derived. The original solution to the circular extrema finding problem was due to LeLann [27] and required $O(n^2)$ message passes. In [28], the algorithm proposed by Chang and Roberts requires $O(n \log n)$ in the average case, and $O(n^2)$ in the worst case. Both solutions were unidirectional. In [29], the algorithm by Hirschberg and Sinclair has $O(n \log n)$ message passes in the worst case, but requires bidirectional communication. In [30], Burns has a slightly better bidirectional $O(n \log n)$ algorithm.

The problem of finding the extrema of any binary function was given by Dobkin and Suri [31], they proposed algorithm for maintaining such extrema in a semi-online model of computation. However, no fully dynamic algorithm for these problems was

previously known. Their algorithm needs the existence of a data structure that, given a value y, then finds the value x among the given inputs that minimize $f(x, y)$. This class of functions includes the closest pair problem, the diameter problem, and many similar geometric optimization problems. Dobkin and Suri called this problem the problem of finding extrema of decomposable functions. The work in [32] proposed a method to solve the problems of the minimum separation among a set of rectangles or higher-dimensional axis-aligned boxes, the extrema distance between a set of points and a set of hyperplanes, and the extrema axis-aligned rectangles or boxes having a long diagonal defined by a pair of points. In [33], David and Elaine presented a method to search for distance function extrema from a point to a curve or a surface. They used geometric operations to find all local extrema. David and Elaine approach is based on hierarchical pruning of the spline model. This type of approach is commonly used to find a global extrema by bounding portions of the geometry with a conservative spatial primitive, such as a sphere or oriented bounding box. Instead of bounding just geometry, their approach also bounds tangent spreads on portions of the geometry and searches to satisfy the zeros of a distance extrema function. The theoretical ground and the mathematical rules for finding the extrema value of a single function f that depends on several independent variables $(x_1, x_2, \ldots, x_n)$ in d-dimensional space are explained in [34]. In this paper, we present self-decomposing general algorithm for finding the extrema of any function whose arguments are stored across a number of distributed databases.

3 INTEGRATION OF DISTRIBUTED DATA

In a distributed setting, a dataset $\mathcal{D}$ is implicitly defined in n explicit databases. We model databases D_is at the i^{th} sites, by a *relation* containing a number of tuples. Each D_i contains set of attributes represented by X_i. For any two databases D_i and D_j, the corresponding sets X_i and X_j may have a set of shared attributes given by S_{ij}. Since an arbitrary number of independent, already existing databases may be consulted for a computation, we cannot assume any data normalization to have been performed for their schemas.

The implicit data set $\mathcal{D}$ with which the computation is to be performed is a subset of the set of tuples generated by a *Join* operation performed on all D_i's. However, the tuples of $\mathcal{D}$ cannot be made explicit at any network site because entire databases, D_i's, cannot to be moved to other sites. The tuples of $\mathcal{D}$, therefore, must remain implicitly specified. This inability to make the tuples of $\mathcal{D}$ explicit is the main problem addressed in the generalized decomposition of global algorithms. To facilitate computations with implicitly specified $\mathcal{D}$, we define a set S that is the union of all the attribute intersection sets S_{ij}, that is,

$$S = \bigcup_{i \neq j} S_{ij}. \tag{1}$$

The set S contains the names of all those attributes that occur in more than one D_i. We define a relation *Shared* which contains all possible tuples that can be enumerated for the attributes in the set S.

3.1 Nature of Data Distribution

The dataset $\mathcal{D}$ consists of a set of tuples where each tuple stores the values of relevant attributes. The distributed nature of such a dataset can lead to two common types of data fragmentation: *horizontal fragmentation* wherein subsets of data tuples are stored at different sites; and *vertical fragmentation* wherein subtuples of data tuples are stored at different sites. Assume that a dataset $\mathcal{D}$ is distributed among n sites containing databases $D_1, D_2, \ldots, D_n$. The individual databases D_i together constitute the implicit global dataset $\mathcal{D}$.

Horizontal Fragmentation: A dataset $\mathcal{D}$ is partitioned into a set of databases $D_1, D_2, \ldots, D_n$ each of which have the same set of attributes X_i, and a subset of tuples of the original dataset $\mathcal{D}$. Each tuple of $\mathcal{D}$ is in exactly one database. The set of shared attributes S is the same as X_i for each database. The union of all databases D_i constitutes the complete dataset $\mathcal{D}$, i.e., $D_1 \cup D_2 \cup \ldots \cup D_n = \mathcal{D}$.

Vertical Fragmentation: A dataset $\mathcal{D}$ is partitioned into a set of databases D_1, $D_2, \ldots, D_n$ where each database contains a subset of the attributes of the original dataset $\mathcal{D}$. Each attribute must be included in at least one database. It is possible for some attributes to be shared (duplicated) across more than one database.

In effect, each D_i is a projection of an implicit global $\mathcal{D}$. Vertically fragmented datasets are more interesting because they provide an opportunity to share knowledge across the participating sites.

3.2 Agent's Decomposition Task and Our Contribution

The objective of an agent is to perform the global computation by communicating with other similar agents at other sites; and each agent performing some computation with its local database. Each agent should be able to decompose the global computation into local computations – in the context of and as constrained by the sharing of attributes across the participating agents – and perform its local part with its own data. Each agent $Agent_i$ in Figure 1 represents a D_i and communicates with similar agents at other nodes to exchange the results of its local computations. The decomposition methodologies discussed here can be seen to reside with each individual agent; and each agent is also capable of initiating and completing an instance of a global computation by either exchanging local results with other agents stationary at their respective sites, or by launching a mobile agent that visits other network sites. In the case of a mobile agent, the decomposition tools and knowledge reside with the mobile agent.

Let us say a result $\mathcal{R}$ is to be obtained by applying a function $\mathcal{F}$ to the implicit dataset $\mathcal{D}$. That is:

$$\mathcal{R} = \mathcal{F}(\mathcal{D}). \tag{2}$$

When the global computation is to find extrema in distributed data components, the value of $\mathcal{R}$ is the tuple at which the given function has the maximum or minimum value across the global data; $\mathcal{D}$ is the database containing the data; and $\mathcal{F}$ corresponds to the implementation of an algorithm for inducing $\mathcal{R}$, from $\mathcal{D}$. Distributed databases used by the agents cannot make explicit the tuples of $\mathcal{D}$, which remain implicit in terms of the explicitly known components $D_1, D_2, \ldots, D_n$. The set S of *Shared* attributes determines what explicit $\mathcal{D}$ would be generated by the individual data components. An implementation of $\mathcal{F}$ in Equation (2) above, for some S, can be engineered by a functionally equivalent formulation given as:

$$\mathcal{R}(S) = H[h_1(D_1, S), h_2(D_2, S), \ldots, h_n(D_n, S)]. \tag{3}$$

That is, a local computation $h_i(D_i, S)$ is performed by agent $Agent_i$ using the database D_i and the knowledge about the attributes *Shared* among all the data sites (S). The results of these local computations are aggregated by an agent using the operation H. However, it may not be possible to decompose a complex computation algorithm into local computations and an aggregator. In this case, we can decompose smaller computational primitive steps of such a complete algorithm and the agent keeps track of the control aspects of sequencing various steps of such an algorithm.

The number and nature and h_i operators and the nature of H would vary with the participating D_is and the set of attributes S among them. Hence, a different set of h-operators would need to be generated by the agent for each new instance of D_i's and S.

Figure 2 shows the process by which the agent would compute $\mathcal{R}$ from the D_is. The component operators of a decomposition (H and h_is), therefore, need to be dynamically determined by the agent for each instance of $\mathcal{F}(\mathcal{D})$ depending on the participating nodes; the attributes contained in their native databases; and the sharing pattern of attributes.

Here, using the decomposition formula above, we present a methodology and design two decomposable algorithms for constructing a function from given discrete data, and finding the *extrema* of any function whose arguments are stored across multiple private distributed databases. This methodology consists of a general model for decomposition, and a set of algorithms for realizing this model. Our algorithms have the capability to decompose their computations to fit the nature of data distribution across the network sites. The objective of the algorithms is to perform function computation tasks for any arbitrary data distribution across the network by exchanging summaries derived from local databases. Our algorithms are tailored for situations in which we do not have closely connected processors. There are multiple processors but they are independent and reside at geographically distant sites.

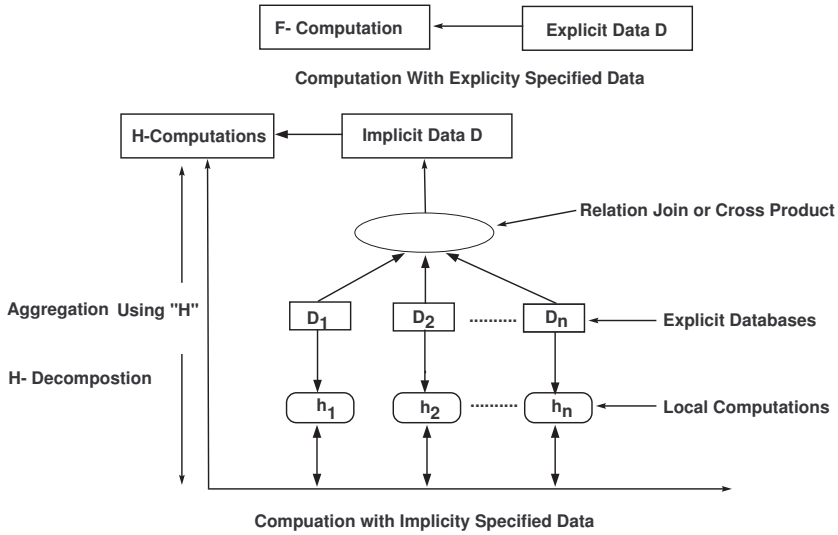

Fig. 2. Computations in explicit vs. implicit data spaces

3.3 Cost Models for Algorithmic Complexity

Traditionally, the complexity of algorithms has been measured in terms of the CPU time and the required memory. This cost model is well-suited for computations on a single computer and the closely-coupled processors model. When a number of loosely networked sites are involved in a cooperative computation the communication cost becomes the overwhelmingly dominant component of the total cost. Complexity for distributed query processing in databases has been discussed in [35]. In our experience with the design and analysis of decomposable network algorithms, we have found that each step of the algorithm must exchange a number of messages for evaluating the various quantitative values. Here and in other similar works [10, 11, 12, 13, 14], we have used cost models involving the number of messages exchanged and reflecting the efficiency of decomposition carried out by the network algorithm. We choose the following two cases for analyzing the complexity of our algorithms.

1. **Stationary Agents Case.** In case of stationary agents, we choose the following two models for analyzing the complexity of our algorithms. In these cost models, we count the number of messages that must be exchanged among all the participating sites in order to complete the execution of the algorithm.

   - *Cost Model 1: Exchanging One Summary per Message (Un-optimized):* One message exchange includes only one local computation request at a time. The

messages are exchanged in a sequential manner, that is, one site is asked for one local computation, the corresponding summary is obtained, and then the request is sent to the next participating database.

- *Cost Model 2: Exchanging All Summaries per Message (Optimized):* One message exchange includes all local computation requests which correspond to all tuples $cond_j$ of S and receive all corresponding summaries in one message.

2. **Mobile Agent Case.** In this case, the complexity can be measured in term of the number of agent hops from one site to other or in term of the number of agent visits to each site. The number of hops/visits is independent of the size of implicit dataset $\mathcal{D}$.

4 DECOMPOSABLE LEAST SQUARES INTERPOLATION

Traditionally, if we have a number n of experimentally determined points, (x_1, y_1, z_1, w_1), (x_2, y_2, z_2, w_2), ..., (x_m, y_m, z_m, w_m), we can construct a linear relation through them in the form

$$W = b + a_1 X + a_2 Y + a_3 Z, \tag{4}$$

where b, a_1, a_2, and a_3 are constants, the right choice of these constants is the one that minimize the sum of the squares of the deviations D.

$$D = \sum_{i=1}^{m} (w_i - (b + a_1 x_i + a_2 y_i + a_3 z_i))^2. \tag{5}$$

Equation (5) represents the deviations between the observed values w_i and the ones $b + a_1 x_i + a_2 y_i + a_3 z_i$ that would be predicted using (4). We calculate the values of b, a_1, a_2, and a_3 from the following partial derivatives:

$$\frac{\partial D}{\partial b} = 0, \frac{\partial D}{\partial a_1} = 0, \frac{\partial D}{\partial a_2} = 0, \text{ and } \frac{\partial D}{\partial a_3} = 0. \tag{6}$$

Using Equations (3), (4), we get the following assistant equations for determining b, a_1, a_2, and a_3:

$$\begin{aligned}
\sum W &= m * b + a_1 \sum X + a_2 \sum Y + a_3 \sum Z, \\
\sum X &= b \sum X + a_1 \sum X^2 + a_2 \sum XY + a_3 \sum XZ, \\
\sum Y &= b \sum Y + a_1 \sum YX + a_2 \sum Y^2 + a_3 \sum YZ, \\
\sum Z &= b \sum Z + a_1 \sum ZX + a_2 \sum ZY + a_3 \sum Z^2.
\end{aligned}$$

In distributed setting, the implicit dataset $\mathcal{D}$ has n attributes B and A_i, where $i = 1, 2, \ldots, m-1$. The objective of our algorithm is to construct a linear function B,

$$B = b + a_1 A_1 + a_2 A_2 + \ldots + a_{m-1} A_{m-1}, \tag{7}$$

where the constants b, a_1, a_2, ..., a_{m-1} can be determined from the assistant equations. In our algorithm, we consider each data tuple in the implicit dataset $\mathcal{D}$ as a point in the d-dimensional space.

4.1 The Algorithm Outlines

In this section, we introduce our algorithm to construct a single function from given discrete data using inter-node communication of local summaries and inferences and obtain the same results that would have been obtained were the computations performed with all the data transferred to a single site.

The Coordinator site will run the following procedures:

1. Call *Count_Computing* procedure to compute the total number of tuples, N_t, in the implicit dataset $\mathcal{D}$
2. Call *Sum_Computing* procedure to compute the summation of each attribute
3. Call *Sum_Product_Computing* procedure to compute the sum of product for each pair of attributes
4. Call *Constant_Determination* to determine the constants.

In the following, we introduce the *Count_Computing*, *Sum_Computing*, *Sum_Product_Computing*, and *Constant_Determination* procedures for computing the total number of tuples in implicit dataset $\mathcal{D}$, the sum of each attribute, the sum of product of each two attributes, and determine constant, respectively from distributed databases.

4.1.1 Count-Computing Procedure

When the tuples of $\mathcal{D}$ are explicitly available in a relation then the count of all its tuples can be easily obtained. For our case of implicity defined $\mathcal{D}$, we can decompose the counting process in such a way that various *local* count requests can be sent to sites of individual $D_k s$ and the responses can then be composed to construct the *total* count for the tuples in implicit dataset $\mathcal{D}$. The decomposition for obtaining the count N_t is as follows:

$$N_t = \sum_l \left(\prod_{k=1}^{n} N(D_k)_{cond_l} \right), \tag{8}$$

where the subscript $cond_l$ specifies a condition composed from the attribute value pairs of the l^{th} tuple of the relation *Shared*, n is the number of participating databases (D_ks), and $(N(D_k)_{cond_l})$ is the count in relation D_k of those tuples that satisfy the condition $cond_l$. Each term in the product is the count of tuples which satisfy condition $cond_l$ in a D_k. The resulting product produces the number of distinct tuples that would be contributed to the implicit *join* of all the D_ks for the condition specified by $cond_l$. The summations in the above expression amount to selecting

each tuple of *Shared* as $cond_l$ and then summing the product terms obtained for each tuple. This expression, therefore, simulates the effect of a *join* operation performed on all the n databases without explicitly enumerating the tuples. A desirable aspect of the above decomposition of N_t is that each local computation $N(D_k)_{cond_l}$ can be translated into an SQL query: *Select count (*) where* $cond_l$ and sent to the site containing database D_k. The pseudocode for computing N_t is shown below:

Count of Tuples in Implicit Space 1. *Local Computation:* The following code will be executed at every local data site D_k

(a) for every shared tuple l do

- Compute $N(D_k)_{cond_l}$, and ship the results back to the Coordinator site

(b) end for

2. *Global Computation:* The following code will be executed at the Coordinator site

(a) for every shared tuple l, compute the total number of tuples that satisfy $cond_l$ from the following relation:

$$N_l = \prod_{k=1}^{n} N(D_k)_{cond_l}. \tag{9}$$

(b) Compute the total number of tuples from the following relation:

$$N_t = \sum_{l} N_l = \sum_{l} \left(\prod_{k=1}^{n} N(D_k)_{cond_l} \right). \tag{10}$$

4.1.2 Sum_Computing Procedure

For an attribute x, we need to compute $\sum_i x_i$ where i is an index over all tuples in implicit dataset $\mathcal{D}$. There are two different cases of x (shared or unshared), each case requires a different way to compute the summation. They are described as follows:

Case 1: $\sum_i x_i$ when $x \in S$: In this case the attribute x belongs to the relation *Shared*. The value $\sum_i x_i$ where i is an index over all tuples in $\mathcal{D}$ can be computed as:

$$Sum = \sum_{l} (x_l * N_l) \tag{11}$$

where l indexes over all the tuples of relation *Shared*, and N_l is similar to Equation (9).

Case 2: $\sum_i x_i$ when $x \notin S$: In this case x is not a shared attribute so it must be reside on a single network site. In this case, the sum of x can be computed using the following procedure:

- For every tuple l in *Shared*,
  - Send a message to each participating site and obtain the average value: $\overline{x_l} = \frac{1}{n}\sum_i x_i$ from the site that contains x.
  - Count the tuples in $\mathcal{D}$ that belong to l.
  - The sum of x can be computed as:

$$Sum = \sum_l (\overline{x_l} * N_l). \tag{12}$$

4.1.3 Sum_Product_Computing Procedure

For a sum of products, we need to compute $\sum_i x_i y_i$ where x and y are two attributes in the implicit dataset $\mathcal{D}$ and i is an index over all tuples in $\mathcal{D}$. Due to the different ways in which attributes x and y may be distributed across the network sites, it turns out that there are *six* different cases, each case requires a different way to compute the sum of products. They are described as follows:

Case 1: $\sum_i x_i^2$ when $x \in S$: In this case x is a shared attribute and the value $\sum_i x_i^2$ where i is an index over all tuples in $\mathcal{D}$ can be computed as

$$SP = \sum_l (x_l^2 * N_l) \tag{13}$$

where l indexes over all the tuples of relation *Shared*, and N_l is similar to Equation (9).

Case 2: $\sum_i x_i^2$ when $x \notin S$: In this case x is not a shared attribute so it must reside on a single network site. For every tuple l in *Shared* we send a message to each participating data site and obtain the average value: $\overline{x_l^2} = \frac{1}{n}\sum_i x_i^2$ from the site that contains x. We also obtain the count of tuples in $\mathcal{D}$ that belong to l. The sum of product can be computed as

$$SP = \sum_l (\overline{x_l^2} * N_l). \tag{14}$$

Case 3: $\sum_i x_i y_i$ when both $x, y \in S$: Since both the attributes are shared attributes, for each tuple l in *Shared* we can form the product of the values taken by x and y attributes in this tuple and multiply it by the count of tuples in $\mathcal{D}$ for which the conditions of this tuple are true. Thus,

$$SP = \sum_l (x_l * y_l * N_l). \tag{15}$$

Case 4: $\sum_i x_i y_i$ when only one attribte is in S: Let us say $x \in S$ and $y \notin S$. The sum of product can be computed as

$$SP = \sum_l (x_l * \overline{y_l} * N_l) \tag{16}$$

where l indexes over all the tuples of relation *Shared*, $\overline{y_l}$ is the average of y for all tuples that belong to l in $\mathcal{D}$ and N_l is similar to Equation (9).

Case 5: $\sum_i x_i y_i$ when both $x, y \notin S$ and x and y reside on different network sites: In this case x and y are not shared and they reside on different sites. For each tuple l in relation *Shared* we obtain $\sum x_l$ and $\sum y_l$ for all tuples that belong to l. Thus the sum of product can be computed as

$$SP = \sum_l \left(\sum(x_l) * \sum(y_l) \right). \tag{17}$$

The sum of product for this case can also be computed as follows:

$$SP = \sum_l (\overline{x_l} * \overline{y_l} * N_l) \tag{18}$$

where l indexes over all the tuples of relation *Shared*, $\overline{x_l}$ and $\overline{y_l}$ are the average of x and y respectively for all tuples that belong to l in $\mathcal{D}$ and N_l is similar to Equation (9).

Case 6: $\sum_i x_i y_i$ when both $x, y \notin S$ and x and y reside on same site: In this case, for each tuple l in relation *Shared* we obtain $\overline{x_l y_l} = \frac{1}{n} \sum_i x_i * y_i$ for all tuples that belong to l. Thus the sum of product can be computed as:

$$SP = \sum_l (\overline{x_l y_l} * N_l). \tag{19}$$

The above six cases cover all possible ways in which the attributes may be distributed across the network. The Coordinator site sends its requests to all the participating data sites and only the sites that have replications will send the response.

4.2 Constant_Determination Procedure

This procedure will be executed at the Coordinator site using the results of the above procedures.

1. Construct the following assistant equations:
   - $\sum B = N_t b + \sum_{i=1}^{m-1} (a_i * \sum(A_i))$, where m is the number of attributes
   - *for* $i = 1$ to $m - 1$
     $\sum BA_i = b \sum A_i + \sum_{j=1}^{m-1} (a_j * \sum(A_i * A_j))$.
2. Solve the constructed assistant equations to determine the constants.
3. Return the relation $B = b + a_1 A_1 + a_2 A_2 + \ldots + a_{m-1} A_{m-1}$.

4.3 Complexity and Analysis

Stationary Agent Case: We give below an expression for the number of messages that need to be exchanged for dealing with the implicit of tuples. Let us say, there are: (1) n relations residing at n different network sites, (2) r tuples in relation *Shared*.

- *Cost Model 1:* The number of messages needed will be the sum of the number of messages required to compute N_t, the number of messages required to compute $\sum x$, and the number of messages required to compute $\sum xy$, each of which will take $n*r$ messages. Thus the total number of messages exchanged will be

$$Exchanged\ Messages = 3n * r. \tag{20}$$

- *Cost Model 2:* In this cost model, values corresponding to all tuples $cond_l$ of S send in one request and receive the summaries in one message. This reduces the number of messages exchanged to n, the same as the number of participating sites. Thus the total number of messages exchanged will be

$$Exchanged\ Messages = 3 * n. \tag{21}$$

The trade-off between the two approaches is that the first one may be considered more secure for transmission over a network because each message contains only very little information about the participating databases. The second alternative requires very few messages but each message contains more information about each database. The above analysis shows that according to the above formulas, in case of One Summary per Message the maximum number of messages which would need to exchange is proportional to the the number of tuples in *Shared* and the number of participating sites, and in case of Exchanging all the Summaries in one Message the maximum number of messages which would need to exchange is proportional to the number of participating sites only.

Mobile Agent Case: This agent has the relation *Shared* stored in it. During a visit to a data site, it computes all local computations for that site. The local results for computing all the counts can be gathered during a single visit to a site. Thus, the mobile agent can compute all terms of assistant equations by visiting each site only once (n hops) and then aggregating the local results.

Assertion 1. The algorithm for constructing a function for $\mathcal{D}$'s attributes in distributed data returns the same results with respect to the algorithm for constructing a function for centralized data attributes.

Proof. It is obvious that the counts N_t, $\sum x$, and $\sum xy$ computed in the distributed case are the same as the counts computed in the centralized case. Then the assistant equations are identical, i.e., we obtain the same results as if we brought all the data together and construct these equations globally. □

4.4 Simulation Results

We have performed a number of tests to demonstrate that the interpolation can be computed in a distributed knowledge environment without moving all the databases to a single site. The following figures show the results in form of graphs that provide a comparative analysis when the algorithm is run using the unoptimized version, i.e. sending one summary per message, and using the optimized version, i.e., sending all the summaries for a particular site in one message.

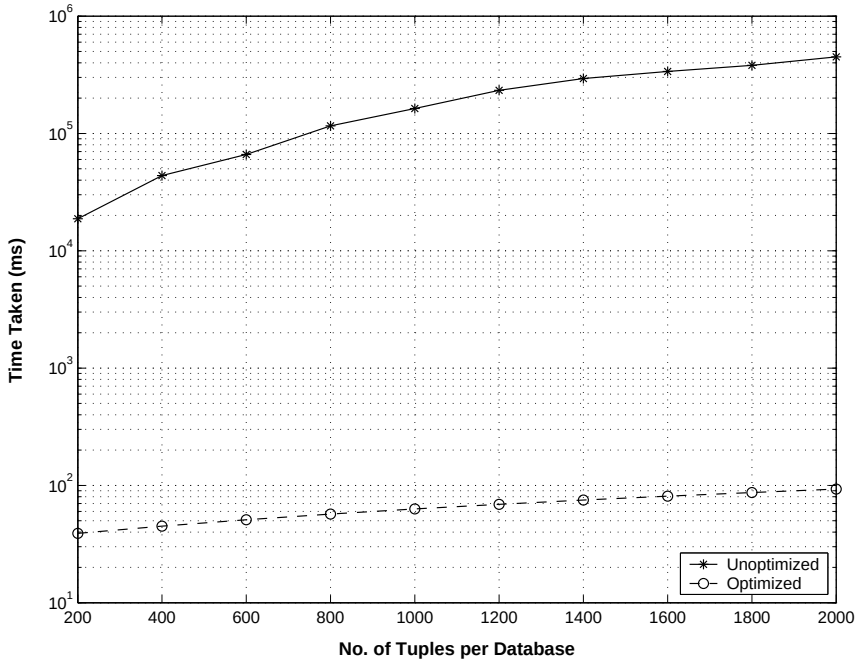

Fig. 3. Time taken to calculate interpolation on distributed databases

Figure 3 shows how the time taken to compute the interpolation in an implicit database $\mathcal{D}$ changes with the size of the individual databases. As we can see, when we exchange one summary per message, the time taken to compute the interpolation varies exponentially as the size of the database increases. However, when we use the optimized method the time taken to compute the interpolation reduces considerably and depends on the number of participating sites.

Figure 4 shows how the number of messages exchanged between the Coordinator site and the remote sites varies with the number of tuples in the database. It can be seen easily that the number of messages exchanged varies with the size of the database when we send one summary per message. The result validates the expression for the total number of messages exchanged as given above. However in

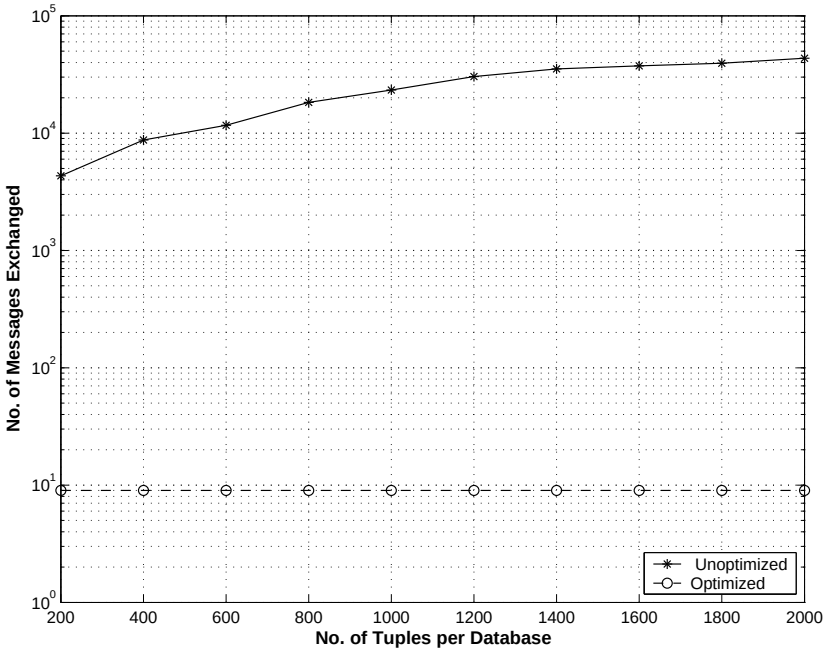

Fig. 4. Number of messages exchanged to calculate interpolation on distributed databases

the optimized version when we receive all the summaries in a single message, the number of messages exchanged was a constant depending upon the total number of participating sites.

5 DECOMPOSABLE FUNCTION OPTIMIZATION

The objective of the second algorithm is to find the extrema (maximum) of a function f that depends on several independent arguments $(x_1, x_2, \ldots, x_d)$ using internode communication of local summaries and inferences and obtain the same results that would have been obtained were the computations performed with all the data transferred to a single site. In our algorithm, we consider each data tuple in the implicit dataset $\mathcal{D}$ as a point in the d-dimensional space and consider the attribute as the function arguments.

Our algorithm consists of two parts; the first part aims to reduce the number of tuples in the relation *Shared*, where every local data site D_i filters the *Shared* relation to the shared tuples that may achieve the maximum value of the function through them. The second part aims to find the candidate point at which the function takes the maximum value at each local data site D_i.

Definition 1. A function $f(A, B, C, \ldots, W)$ has an *extrema* (maximum value) at a point $(a, b, c, \ldots, w)$, if the value of the function $f(A, B, C, \ldots, W)$ at all points that sufficiently close to this point is less than the value of $f(a, b, c, \ldots, w)$.

5.1 Algorithm Outlines

Data structure: A table called *Max* is maintained at the Coordinator site. It has one column for every attribute to store the candidate value for extrema, and one column for the corresponding function value.

Reduce the number of tuples in the relation Shared as follows:

1. *Local computation:* The following code will be executed at every participating site to find the candidate shared tuples.
   - *for* every shared tuple l *do*
     - Select all tuples that belong to l.
     - Compute the function value at every selected tuple (consider the unknown attributes at every D_i as constants).
     - Sort the values in decreasing order.
     - Return the shared tuple that corresponds to the maximum value to the Coordinator site.
2. *Global computation:* At the Coordinator site, we generate from the returned tuples a new relation called *MaxShared*, that has a number of tuples much less than the original *Shared* relation.

Finding the extrema point:

1. *Local Computation:* We find the corresponding unshared values of every tuple in *MaxShared*
   - *for* every tuple j in *MaxShared do*
     - Select all tuples that corresponds to j,
     - *if* the number of selected tuples = 1,
       - Return the unshared attributes value corresponding to this tuple to Coordinator site.
     - *else*
       - Compute the function value at each tuple (consider the unknown attributes as constants),
       - Sort the function values in decreasing order,
       - Return the unshared attributes values corresponding to the maximum value to the Coordinator site.
2. *Global Computation:* The Coordinator site will execute the following steps to find the tuple at which the function has maximum value:

- Update the *Max* table for every tuple in *MaxShared*,
- Compute the function value f_{val} for each row in *Max* table.
- Return the tuple corresponding to the maximum value of f_{val}.

End algorithm.

5.2 Complexity and Analysis

Stationary Agent Case: We give below an expression for the number of messages that need to be exchanged for dealing with the implicit tuples. Let us say there are:

1. n relations, residing at n different network sites,
2. r tuples in relation *Shared*,
3. $\acute{r}$ tuples in relation *MaxShared*.

- *Cost Model 1:* We send one message to each participating data site to find the shared tuple at which the function takes on the maximum value. We send one message to each data site to find the unshared attributes values where the function takes on the maximum value for each tuple $\acute{r}$. Thus the total number of messages exchanged will be

$$Exchanged\ Messages = n + n * \acute{r} = n(1 + \acute{r}). \tag{22}$$

- *Cost Model 2:* In this cost model, we send n messages for generating the relation *MaxShared*. We send one message to each data site to find the unshared attributes values where the function takes on maximum value for all tuples $\acute{r}$. Thus the total number of messages exchanged will be

$$Exchanged\ Messages = 2 * n. \tag{23}$$

The above analysis shows that according to the above formulas, in case of One Summary per Message the maximum number of messages which would need to exchange is proportional to the the number of tuples in *MaxShared* and the number of participating sites, and in case of Exchanging all the Summaries in one Message, the maximum number of messages which would need to exchange is proportional to the number of participating sites only.

Mobile Agent Case: This agent has the relation *Shared* stored in it. During a visit to a data site, it computes the function value for all shared tuples and keeps the shared tuple that satisfies the maximum value of the function in its mind. Once all the sites have been visited, the agent generates the relation *MaxShared* according to the tuples that have been kept. The mobile agent would need to visit each site another time for computing the global value of the function for each tuple $\acute{I}$. Thus the total number of visits (hops) will be

$$Number\ of\ hops = 2n. \tag{24}$$

Assertion 2. If all subsets of a point P cannot maximize a function f, where the rest of P is constant, then P cannot maximize f.

Proof. Assume $f = x + y + z$ has the maximum value at the point $p_0 = (x_0, y_0, z_0)$, and p_0 maximizes f but all of its subsets cannot maximize f. Since p_0 maximize f, then $f(x_0, y_0, z_0) > f(x, y, z)$ for all points (x, y, z), i.e.,

$$x_0 + y_0 + z_0 > x + y + z. \tag{25}$$

If z is constant, then $x_0 + y_0 + z > x + y + z$ which is contradiction. □

Assertion 3. The algorithm for finding the extrema from distributed data returns the same results with respect to the algorithm for finding the extrema from centralized data.

Proof. Using the same assumptions as in Assertion 2, if x is constant, then

$$x + y_0 + z_0 > x + y + z. \tag{26}$$

If y is constant, then

$$x_0 + y + z_0 > x + y + z. \tag{27}$$

If z is constant, then

$$x_0 + y_0 + z > x + y + z. \tag{28}$$

By adding Equations (26), (27), and (28), we find that $x_0 + y_0 + z_0 > x + y + z$, i.e., we obtain the same results as if we brought all the data together and compute the extrema. □

Since the maximized operation is opposite of the minimized operation, we can modify our algorithm to find the extrema (minimum) value of a function by inverting the maximized instructions to minimized instructions.

5.3 Simulation Results

We have performed a number of tests to demonstrate that the extrema can be computed in a distributed knowledge environment without moving all the databases to a single site. The following figures show the results in form of graphs that provide a comparative analysis when the algorithm is run using the unoptimized and optimized versions.

Figure 5 shows how the time taken to compute the extrema in an implicit database $\mathcal{D}$ changes with the size of the individual databases and the number of tuples in *MaxShared*. As we can see, when we exchange one summary per message, the time taken to compute the extrema varies exponentially as the size of the database increases. However, when we use the optimized method the time taken to compute the extrema reduces considerably and depends on the number of participating sites.

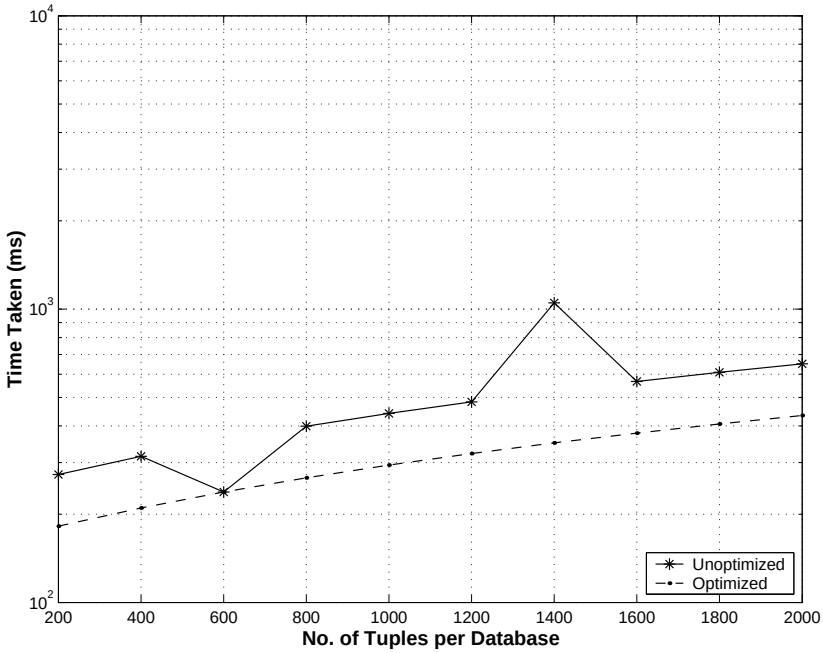

Fig. 5. Time taken to calculate the extrema on distributed databases

Figure 6 shows how the number of messages exchanged between the Coordinator site and the remote sites varies with the number of tuples in *MaxShared*. It can be seen easily that the number of messages exchanged varies with the size of the database and the number of tuples in *MaxShared* when we send one summary per message. The result validates the expression for the total number of messages exchanged as given above. However, in the optimized version when we receive all the summaries in a single message, the number of messages exchanged was a constant depending upon the total number of participating sites.

6 CONCLUSION

We have demonstrated that agents can perform integration of arbitrarily distributed data and knowledge for performing computations. Tasks such as computing extrema and running interpolation can be computed by appropriately coordinating agent actions. These actions are self-determined and self-controlled by the agents in response to the varying sets of participating agents and arbitrary overlaps in the local datasets. Also, for simple arithmetic computations, the number of messages to be exchanged among the n participating agents does not exceed the order of n. This is very significant because it gives us the scalability required for handling large databases. The number of tuples at individual network sites may keep on increas-

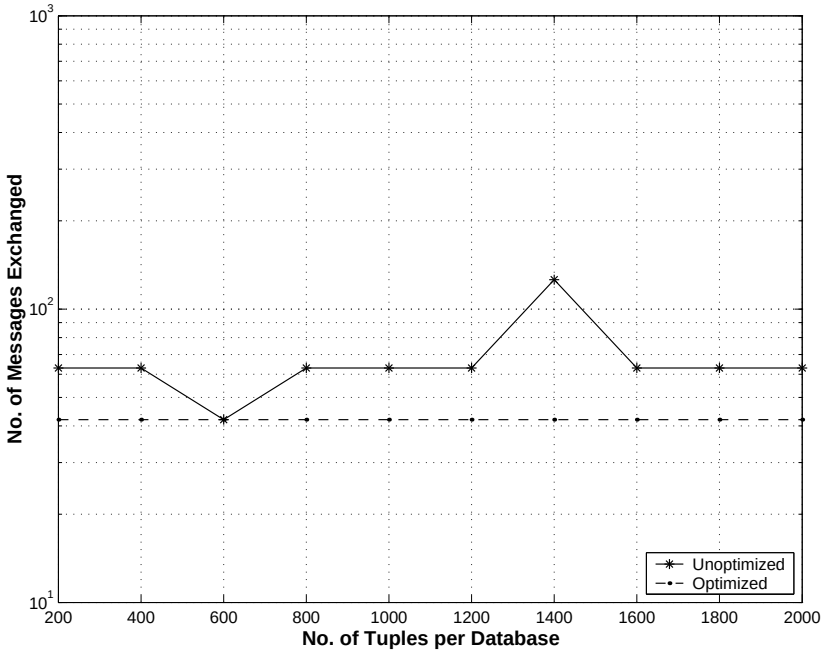

Fig. 6. Number of messages exchanged to calculate the extrema on distributed databases

ing but the number of messages that need to be exchanged among the agents for a global computation remains constant. We have demonstrated the adaptability of the extrema finding algorithm, and interpolation algorithm. We have shown the complexity of performing these computations in terms of messages that need to be exchanged among the stationary agents for performing these computations. We have also analyzed the number of visits that a mobile agent would need to make to each site for completing the global computation. One very significant contribution of these results is that many tasks can be performed on a number of databases residing at different network sites by agents without having to move the databases to a single site and the communication cost among the performing agents is also very low.

REFERENCES

[1] Curtis-Maury, M.—Shah, A.—Blagojevic, F.—Nikolopoulos, D.—de Supinski, B.—Schulz, M.: Prediction Models for Multidimensional Power-Performance Optimization on Many Cores. In Proceedings of the 17$^{\text{th}}$ International Conference on parallel architectures and compilation techniques, ACM New York, NY, USA, 2008, pp. 250–259.

[2] Park, S.—Jiang, W.—Zhou, Y.—Adve, S.: Managing Energy – Performance Tradeoffs for Multithreaded Applications on Multiprocessor Architectures. In Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, ACM New York, NY, USA, 2007, pp. 169–180.

[3] Wang X.—Ziavras, S.: Performance-Energy Tradeoffs for Matrix Multiplication on FPGA-Based Mixed-Mode Chip Multiprocessors. In Proceedings of the 8th International Symposium on Quality Electronic Design 2007, pp. 386–391.

[4] Jin, R.—Yang, G.—Agrawal, G.: Shared Memory Parallelization of Data Mining Algorithms: Techniques, Programming Interface, and Performance. IEEE Trans. Knowledge Data Eng., Vol. 17, 2005, No. 1, pp. 71–89.

[5] Schwarz, T.—Iofcea, M.—Grossmann, M.—Honle, N.—Nicklas, D.—Mitschang, B.: On Efficiently Processing Nearest Neighbour Queries in a Loosely Coupled Set of Data Sources. In ACM International Workshop on Geographic Information Systems, Washington, DC 2004, pp. 184–193.

[6] Aho, A. V.—Hopcroft, J. E. Ullman, J. D.: The Design and Analysis of Computer Algorithms. Addison Wesley, MA 1974.

[7] Aki, S. J.: Parallel Computation: Models and Methods. Prentice Hall, New Jersey 1997.

[8] Barbosa, V. C.: An Introduction to Distributed Algorithms. MIT Press 1996.

[9] Jaja, J.: An Introduction to Parallel Algorithms. Addison Wesley Publishers 1992.

[10] Khedr, A. M.: Nearest Neighbor Clustering Over Partitioned Data. Computing and Informatics, Vol. 30, 2011, No. 6, pp. 1001–1026.

[11] Khedr, A. M.: Learning k-Classifier from Distributed Databases. Computing and Informatics, Vol. 27, 2008, pp. 355–376.

[12] Khedr, A. M.—Salim, A.: Decomposable Algorithms for Finding the Nearest Pair. J. Parallel Distrib. Comput., Vol. 68, 2008, pp. 902–912.

[13] Khedr, A. M.—Bhatnagar, R.: A Decomposable Algorithm for Minimum Spanning Tree. Distributed Computing Lecture Notes in Computer Science, Vol. 29, No. 18, pp. 33–44, Springer-Verlag Heidelberg 2004.

[14] Khedr, A. M.—Bhatnagar, R.: Agents for Integrating Distributed Data for Complex Computations. Computing and Informatics, Vol. 26, No. 2, 2007, pp. 149–170.

[15] Bhatnagar, R.—Srinivasan, S.: Pattern Discovery in Distributed Database. Proceedings of the 14th National Conference of AAAI 1997, pp. 508–513.

[16] Chia, M. H.—Neiman, D. E.—Lesse V. R.: Poaching and Distraction in Asynchronous Agent Activities. ICMAS 1998, pp. 88–95.

[17] Durfee, E. H.—Corkill, V. R.: Coherent Cooperation Among Communicating Problem Solvers. IEEE Transactions on Computers, Vol. 36, 1987, No. 11, pp. 1275–1291.

[18] Huhns, M. N.—Singh, M. P.—Ksiezyk, T.: Global Information Management via Local Autonomous Agents. Morgan Kaufmann Publishers, 1997.

[19] Toomer, G. J.: Hipparchus. In Dictionary of Scientific Biography, C. C. Gillispie and F. L. Holmes (Eds)., Scribner, New York 1978, Vol. XV, pp. 207–224.

[20] Grevara, G. J.—Udupa, J. K.: An Objective Comparison of 3-D Image Interpolation Methods. IEEE Trans. Med. Imag., Vol. 17, 1998, pp. 642–652.

[21] Evans, W. D.—Opic, B.—Pick, L.: Real Interpolation with Logarithmic Functors. Journal of Inequalities and Applications Vol. 7, 2002, No. 2, pp. 187–269.

[22] Sol, J.—Salembier, P.: Quadratic Interpolation and Linear Lifting Design. EURASIP Journal on Image and Video Processing, Vol. 27, 2007, ID 37843.

[23] Chang, J. Z.—Allebach, J. P.—Bouman, C. A.: Sequential Linear Interpolation of Multidimensional Functions. IEEE Transactions on Image Processing, Vol. 6, No. 9, 1997, pp. 1231–1245.

[24] Narayanaswami, C.: Efficient Parallel Gouraud Shading and Linear Interpolation over Triangles. Computer Graphics Forum, Vol. 14, 1995, No. 1, pp. 17–24.

[25] Zheng, Y.—Doerschuk, P. C.—Johnson, J. E.: Symmetry-Constrained 3-D Interpolation of Viral X-Ray Crystallography Data. IEEE Transactions on Signal Processing, Vol. 5, 2000, No. 1, pp. 2933–2935.

[26] Courtois, C.—Denuit, D.—Bellegem, S. V.: Discrete S-Convex Extrema, with Applications in Actuarial Science. Proceedings of the 3rd Actuarial and Financial Mathematics Day, Brussels 2005, pp. 35–46.

[27] Lelann, G.: Distributed systems – Towards a formal approach. Information Processing 77, Elsevier Science, New York 1977, pp. 155–160.

[28] Chang, E.—Roberts, R.: An Improved Algorithm for Decentralized Extrema Fmding in Circular Configurations of Processes. Commun. ACM 22, May 1979, pp. 281–283.

[29] Hirschberg, D. S.—Sinclair, J. B.: Decentralized Extrema-Finding in Circular Configurations of Processors. Commun. ACM 23, Nov. 1980, pp. 627–628.

[30] Burns, J. E.: A Formal Model for Message Passing Systems. Tech. Rep. 91, Computer Science Dep., Indiana Univ., Bloomington, May 1980.

[31] Dobkin, D.—Suri, S.: Maintenance of Geometric Extrema. J. Assoc. Comput. Mach., Vol. 38, 1991, pp. 275–298.

[32] Eppstein, D.: Dynamic Euclidean Minimum Spanning Trees and Extrema of Binary Functions. Discrete & Computational Geometry, Vol. 13, No. 1, 1995, pp. 111–122.

[33] Johnson, D. E.—Cohen, E.: Distance Extrema for Spline Models Using Tangent Cones. Proceedings of the Conference on Graphics Interface 2005, pp. 169–175.

[34] Vegodsky, M.: Mathematical Handbook. Mir Publishers, Moscow 1971.

[35] Wang, C.—Chen, M.: On the Complexity of Distributed Query Optimization. IEEE Transactions on Knowledge and Data Engineering, Vol. 8, 1996, No. 4, pp. 650–662.

Ahmed M. Khedr received his B. Sc. degree in Mathematics in June 1989 and the M. Sc. degree in the area of optimal control in July 1995 both from Zagazig University, Egypt. In July 1999 he received his M. Sc. and in March 2003 he received his Ph. D. degree both in area of Computer Science and Engineering from University of Cincinnati, Ohio, USA. From March 2003 to January 2004, he was a research assistant professor at ECECS Department University of Cincinnati, USA. From January 2004 to May 2009, he worked as assistant professor at Zagazig University, Egypt, from September 2009 September 2010 he worked as associate professor at the Department of Computer Science, college of computers and Information Systems, Taif University, KSA, and from September 2010 till now he is working as associate professor at the Department of Computer Sciences, college of science, Sharjah University, UAE. In June 2009, he awarded the State Prize of Distinction in Advanced Technology. He has coauthored 40 works in journals and conferences relating with optimal control, Wireless Sensor Networks, Distributing Computing, and Bioinformatics.