A DATA FLOW IMPLEMENTATION OF AGENT-BASED DISTRIBUTED GRAPH SEARCH

Imad Hamchi, Mathieu Hoarau, Antoine Fillinger[1], Nicolas Crouzier, Lukas Diduch, Martial Michel[2], Vincent Stanford
The National Institute of Standards and Technology 100 Bureau Drive, Gaithersburg Maryland 20899 U.S.A
[1]Dakota Consulting, Inc. Silver Spring, Maryland
[2]Systems Plus, Inc. Rockville, Maryland
Imad.Hamchi@nist.gov, et al.

ABSTRACT

Biological ants organize themselves into forager groups that converge to shortest paths to and from food sources. This has motivated development a large class of biologically inspired agent-based graph search techniques, called Ant Colony Optimization, to solve diverse combinatorial problems. Our approach to parallel graph search uses multiple ant agent populations distributed across processors and clustered computers to solve large-scale graph search problems. We discuss our implementation using the NIST Data Flow System II, and show good scalability of our parallel search algorithm.

KEY WORDS
NIST Data Flow System, Ant Colony Optimization (ACO), Parallel Distributed Processing, combinatorial optimization, parallel graph search.

1. Introduction

Parallel optimization algorithms open larger problems in important areas to solution, but have historically been difficult to implement. However, many of the difficulties can be obviated using mature parallel processing toolkits. Since 1997 we have been developing such a middleware layer for data acquisition and transport in distributed sensing environments known as the NIST Data Flow System Version II (NDFS-II) [1]. Because our distributed sensing environments levy requirements to process multiple signal processing and pattern classification algorithms in realtime, we extended our data flow system to provide necessary logical synchronization of multiple data channels. The usual blocking vs. non-blocking, and order and fairness guarantees provided by Message Passing Interface (MPI) can be selected by the applications programmer. Our middleware layer also provides transport and any necessary remote or local stream duplication. The data are transported through the basic structure of flows, which are buffered queues operating under a publish-subscribe mechanism. It also provides a peer-to-peer discovery mechanism, provides fault tolerance, graphical deployment across computing clusters, control, and status monitoring for complex computing environments.

Increasing availability of multicore processors in low-cost commodity computers and interest in graph search problems for diverse applications including network routing, social networking, and scheduling led us to investigate parallelism using distributed agent-based search algorithms.

Ant Colony Optimization (ACO) is a computational metaheuristic referring to a class of such algorithms first advanced by Marco Dorigo [2]. It uses simple ant-like agents to collaboratively find optimal, or near optimal solutions by laying pheromones down on the search space graph. The fact that the agents communicate with the search space rather than directly with each other made this algorithm attractive for parallel implementation with manageable communication overhead. We discuss our parallel computing layer, its application to ACO, and scaling results below.

Strategies of parallelizing ACO algorithms can be classified into “coarse grained” and “fine grained” categories. Parallelism at the colony level, the “coarse grained” approach, helps to obtain better scalability and reduce the communication overhead. For example, the “fine grained” parallel Ant System implementation proposed by Bolondi and Bondanza [3] consists of assigning one ant to one processor. Using the C/PVM programming environment Talbi et Al [4] implemented such an algorithm for the Quadratic Assignment problem. Souto et al [5] used the same approach with MPI to their “reconstruction of chlorophyll concentration profile in offshore ocean water” application. The speed up value obtained for 3, 5 and 15 processors with a population of 90 ants is 2.65, 3.74 and 6.38 respectively. Bullnheimer et al [6] proposed a “coarse grained” parallelization method where the pheromone matrix is exchanged every fixed number of iterations between the subpopulations. The speed up result obtained for 25 nodes with a population of 500 ants is 20. Pierre Delisle and al [7] presented a shared memory parallel implementation of ACO based on OpenMP that is applied to an industrial scheduling problem. The speedup

factor obtained with a number of ants set to 1000 and 16 processors is 5.45.

2. Parallel ACO using the NDFS II

Parallel processing: the NIST data flow system

The NDFS-II is a general data-abstract distributed computing framework with the following key features:

- Point-to-point data communication.
- High throughput, low latency, and low data-overhead.
- Thread safe.
- Dynamic process management.
- Uses TCP and sockets for data integrity.
- Available for C, C++ and Java.
- Crash-resilient server crossbar information exchange.
- Clients exchange data using data-flows.
- Flows are accessed by name and group, rather than by the client node ID and where they run.
- Flows are data abstract: no simple or complex data-types defined beside the mechanism to embed any data within a flow.
- The data abstraction facilitates use of meta-data within flows, which in turn make it possible to send embedded command and control data.
- Serialized data-types can be simply made by a memory representation within a flow containing an amalgam of network transparent data embedded with reconstruction meta-data.
- User created flows that can therefore use 3rd party tools and libraries to work on the data (e.g. Matlab or GNU Octave) transmitted on a distributed node.
- Send and receive operations can be blocking or non-blocking, with a poll mechanism available for non-blocking operations.
- Data block synchronization can be one of "Exact Block Stamp Match", "Tolerant Time stamp Match", or "Overlay Timestamp Match" [8].
- Rendezvous functionality available for flow synchronization.
- The topology of the graph is embedded within the flows by use of their data types and groups.

It is well designed for data centric applications requiring low communication overhead, high throughput, and low latency data transfer, for use in client-server, peer-to-peer, clustered and N-tiered architectures. It has been successfully used in master-slave and producer-consumer applications. Thanks to its high emphasis on data availability via point-to-point communication, it can be extended to work with distributed hash tables. It can also facilitate data distribution and data balancing, with extensions for automatic load and data balancing in plan.

Networked sensor management in NDFS-II

The earlier NDFS-I was exclusively used in data management for networked sensors. Multiple sensors were directly connected to their own data recording devices, and the data were made available at device clock rates as network transparent flows which could be consumed by multiple clients to process the sensor data under a pull model [9]. The NDFS-II was extended to use more resilient distributed computing features while enabling more sensors to be managed. NDFS-II has been extensively used in multimodal data gathering for research communities worldwide. As described in [1], the NIST Automatic Meeting Recognition Project used seven High Definition (HD) video cameras, twenty-four individual microphones and four 64-channel microphone arrays attached to a network of thirteen hosts to record data to disk, synchronized it, and made it available as flows, for live processing in a review station, routing to recognition clients, and preservation as a research archive of standard reference data.

NDFS-II features for parallel processing

The NDFS-II was originally designed to support the development of real-time pervasive applications by providing distributed sensor data acquisition, transport, and processing capabilities [10]. We soon found that the features for distributed sensor stream transport are readily adaptable to more general parallel processing problems addressed as large-grain data flow graphs. In this system, distributed applications are represented as data flow graphs with computational client nodes as standalone processes working together by providing and/or consuming data flows from sensors or other client nodes.

The flows supported by NDFS-II are network-transparent and accessed using their name and type rather than their location on the network. Thus if two computational components need to perform a task by exchanging data, the source code used to program this operation within the components is the same whether they run together on the same host, or are distributed on multiple hosts and processor cores.

The data flow metaphor for data transport between process-level nodes is well designed for parallel and distributed computing. It transfers data blocks among computational nodes that execute processing pipelines or branch parallelism in applications. In the case of pervasive computing applications, the data transported are often audio or video streams. But the data flow core library is not specialized in multimedia data; rather it works using the concept of data blocks. These blocks are managed by flows that can be specialized to offer pre- and post-processing of the data depending on the data type they transport (audio, video, matrices, etc). The flow concept and the way it was implemented in the middleware facilitate the process of creating new flow types specialized to han-

dle dedicated data types. This process is also simplified by using the flow generator, a GUI tool that generates flow templates. Therefore NDFS-II is well suited for a range of applications beyond pervasive computing data acquisition.

The second generation NDFS is multi-platform, running on Linux, Mac OS X, and Windows, thanks to the use of cross-platform libraries. The Adaptive Communication Environment (ACE) library is used to provide the low level functionalities needed by the core of NDFS-II. The Qt graphical library from Trolltech provides the graphical components enabling us to develop platform independent GUIs, such as the Control Center which is used to create, control and monitor distributed applications from a single host. The NDFS-II core and both of these libraries are implemented in C++, and thus provide the performance required for high data throughput with low processor requirements.

This middleware offers dynamic capabilities, as any host can join or leave an existing NDFS-II network at any time. In the same manner, client nodes running on a host belonging to the NDFS-II network can become part of an application by providing or consuming flows that represent streaming data, or leave a running application by unsubscribing. In order to join a data flow network individual hosts need to run an NDFS-II server that is able to dynamically discover its peer servers to synchronize its database specifying which resources (flows) are available and where they are located. Computational components (clients) can then join the network by connecting to the server on their host and subscribe to flows published by other clients. The server will execute the necessary control operations to connect the clients through the appropriate data flows.

In order to increase the robustness of the system and optimize the data transport rates, clients are not directly connected to one another, but instead use duplicators for point-to-point communication. Client crashes are handled by duplicators: if two clients consume the same flow from a single producer, and one consumer crashes it won't affect the remaining consumer because the duplicator simply stops routing data to the failed client. If this consumer is also a producer, any consumer having subscribed to any of its flow(s) will no longer receive those data.

The use of the duplicator not only increases robustness but also minimizes the data transport bandwidth. Duplicators use shared memory to allow concurrent access to data blocks among client nodes within a machine, and use TCP/IP when data are exchanged among client nodes located on several machines. They create peer-to-peer data links between clients that do not go through a central server. The system also avoids unnecessary data duplication: as all clients access the same shared memory for a dedicated flow within a machine, and if the data blocks need to be transported to a different host, they are only sent once and then duplicated at the destination, thus reducing the network load even if there are several consumers on the remote host, as shown in figure 1.

The data transport can be affected by irregular network conditions or client consumption rates. In order to smooth these effects, a hybrid push/pull mechanism was employed. There is one data queue per flow within a client, managed in its own thread, to avoid blocking the program for unnecessary reasons. When a producer sends a data buffer, it is enqueued rather than being sent immediately. The flow thread then dequeues the data to the shared memory when notified by the flow duplicator. Similarly, when a data buffer is ready for consumption in the shared memory, the flow thread of the consumer is notified and can then retrieve and push the data in its internal queue. The consumer can then pull the data from the queue for smooth playback of streaming media when enough data has arrived. The irregularity in transport can be compensated by the queue buffering the data. It is also an efficient way to distribute data.

Data queue behavior and buffer sizes can be customized to address the needs of different applications. A queue can be configured as either blocking if all data must be processed, or non-blocking if data loss can be tolerated. A real-time application may be able tolerate losing some data: in that case, the queue can be customized to drop the newest, or oldest, data block when it is filled. Other types of applications, such as pipelines to process data files, cannot not tolerate data loss: in which case the queue can be made blocking. In order to handle files, a synchronization mechanism was introduced to make sure that the transfer only starts when all the required clients establish communication through the flows, thereby preventing any data loss.

The NDFS-II API abstracts the transport mechanism, providing methods for clients to join and leave the network at any time, and create flows based on their type for publication and subscription. Data blocks of an output flow can be filled with user data in several ways ranging from direct access of the memory block to the use of dedicated methods to fill out the block. Symmetrically user data stored in a data block from an input flow can be read using direct access to memory or using an iterator on the data block.

An object oriented C++ API is provided, but other language bindings have also been developed, so it is possible to use the system from Java, Matlab and GNU Octave programs. Matlab and Octave provide a powerful set of operators that can be used in NDFS-II data flow graphs for signal processing or pattern classification. For example, one can create a pipeline to handle operations on matrices where each client performs a specific operation on the matrix before passing it to the next.

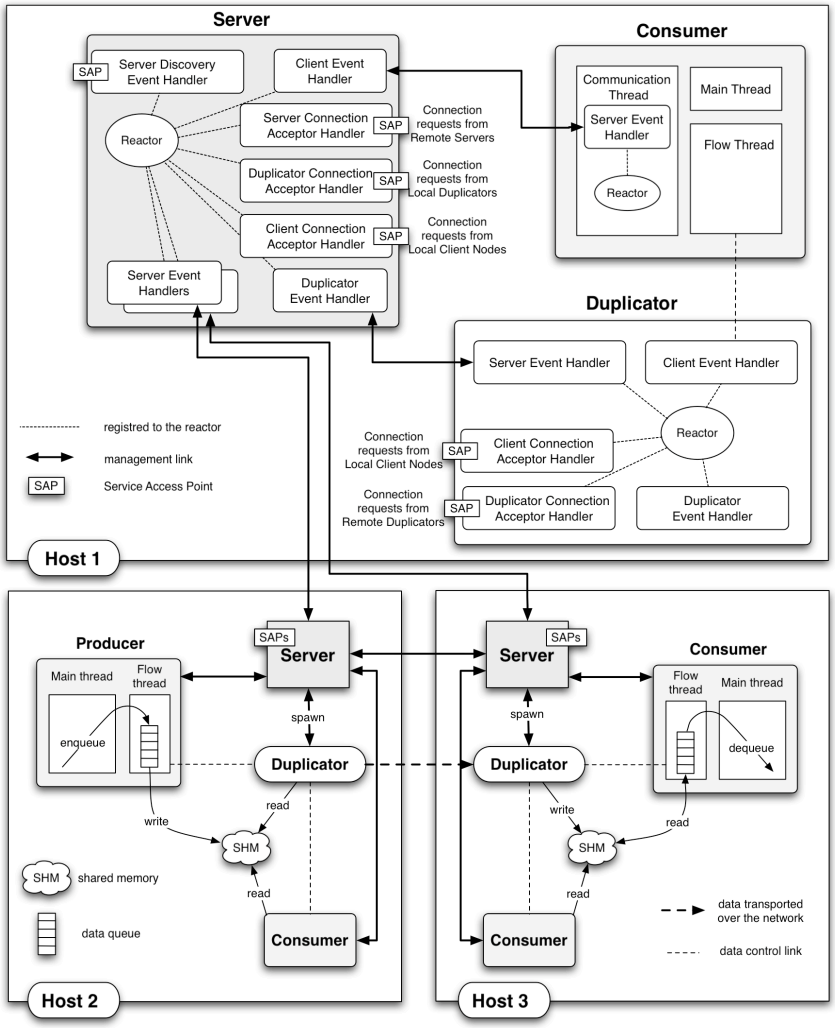

Figure 1. Servers exchange control messages to run clients, and mediate point-to-point data transfer. Duplicators manage shared memory access between clients within a machine and use TCP to send data over the network.

Complex behavior arising in natural ant colonies inspires agent-based search methods

Physical and biological systems composed of many simple agents interacting under simple rules show complex emergent behaviors, or phase transitions, with increasing numbers of agents. Entomologist Paul Grassé first proposed the idea that foraging insect groups, like termites, used pheromone markings to guide other members to food sources, or places of collaborative work like nest construction [11]. Beekman, and her colleagues observed that Pharaoh's Ants (*Monomorium pharaonis*) colonies showed phase transitions from random foraging by individual ants in small numbers, to collaborative patterns for larger numbers [12]. These phase transitions were found to exhibit hysteresis comparable to many physical systems undergoing first-order phase transitions.

Sumpter and Pratt made more detailed analyses of differential equations that model trail maintenance by ant populations which predict transition between search states in ant populations like those observed in the natural ants [13]. Moreover, collaboratively discovered paths are short relative to the average found by individuals in the stochastic search phase. Importantly, the dynamics of phase change described by Beekman et al. imply that searches with too few ant agents will never cohere into efficient social foraging patterns. Thus populations below the social coherence threshold for a given problem space search stochastically, and may take very, very, long times to find an efficient solution. So the ability to operate larger collaborating populations using parallel methods is important for larger graph search problems. Figure 2 shows a graph search problem and distribution of ant foragers for two populations, one too small for the phase transition to collaborative search to occur, and one large enough to achieve phase transition.

Digital Ant Colony Optimization: A biologically inspired agent-based graph search method

We investigated widely studied agent models of ants in the setting of our parallel-distributed processing middleware, the NDFS II, for agent based graph search. This framework allows us to employ multicore processors, and distributed systems to explore much larger problem spaces using agent-based algorithms like ACO than is feasible with single processor implementations. The parallelization method uses the stygmergic markings from multiple agent populations running on many processors and host nodes with a periodic rendezvous/update cycle mediated by a master node.

Work by entomologists including Grassé inspired Marco Dorigo and numerous collaborators to develop agent-based simulations that capture important observed dynamical aspects of biological ant colony foraging [14]. This model of natural ant colonies is composed of numerous simple digital agents that stochastically extend search paths through problem graphs, and interact through stygmergic markings on the arcs they travel. This allowed the study of the optimality and performance properties of these systems [15].

Dorigo and others investigated a variety of pheromone update rules, and constraints on stochastic solution generators that enable complex combinatorial problems to be solved. These ACO system variants included ant-density, ant-quantity, and ant-cycle, min-max, and best-so-far pheromone update methods. Problems that have been successfully addressed include the Traveling Salesman Problem (TSP), network routing, sequence matching, alignment, and scheduling problems. Many of these are NP-hard problems, unlike the shortest path problems with polynomial time solutions that biological ants solve. Dorigo's *Min-Max* method is described below. For our experiments we used the Min-Max method, except the best-so-far update rule. This retains a greater physical realism and facilitates the study of the phase change described by Beekman et al. because ant-agents lay pheromones only on their path of travel.

Summary of the Dorigo MIN-MAX ACO algorithm

One of the most widely studied variants of ACO is the Dorigo MIN-MAX system [16], which we describe as follows. The ACO systems can generate solutions to combinatorial problems specified at the highest level of abstraction as $\{\Sigma, f(S), \Omega(S)\}$, with a solution set Σ, an objective function $f(S)$, and a feasibility constraint $\Omega(S)$. Each solution $S \in \Sigma$ is a state sequence $\{s1, s2, \ldots s_{N(s)}\}$ that satisfies the constraint $\Omega(S)$, and its merit is $f(S)$.

Ant solution construction: Ants are initially situated relative to a particular graph search problem. Then partial solutions generated by each ant are stochastically extended into the adjacent nodes from its present position, if they are feasible under $\Omega(S_k)$ as follows.

Until ($S \in \Sigma$) {

Extend the state sequence $\{s1, s2, \ldots s_k\}$ to include s_{k+1} by finding all feasible successors of s_k on its adjacency list $adj(s_k)$ permissible under $\Omega(s_k)$, and selecting randomly among them according to the probabilities computed as:

$$P(s_{k+1} \mid s_k) = \frac{\tau(s_k, s_{k+1})^{\alpha}}{\sum_{s_{j+1} \in \Omega(adj(s_k))} \tau(s_k, s_{j+1})^{\alpha}}$$

if $\Omega(adj(S_k))$ is the empty set, then the ant is teleported back to the nest and its partial solution is discarded.

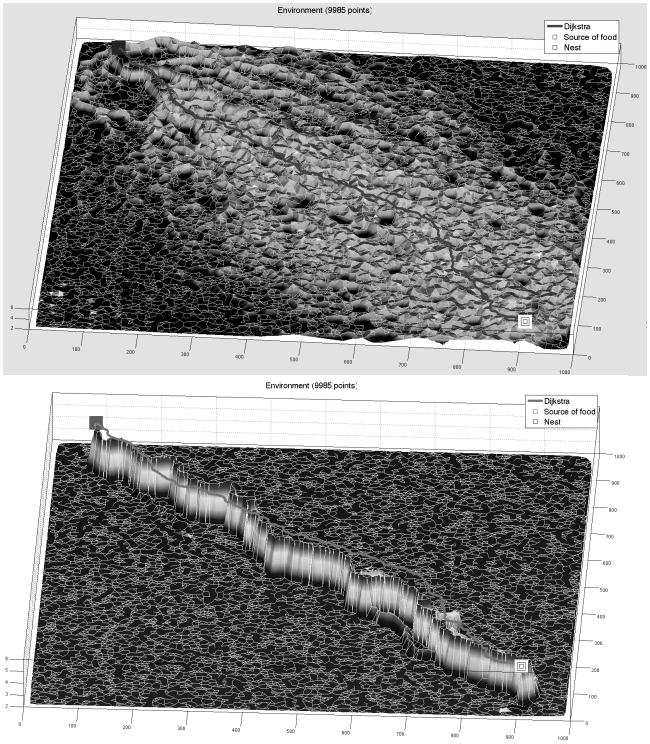

Figure 2. The size of ant forager populations relative to the size of the search space determines if the foragers will explore the search space independently and stochastically (*top*), or collaborates to find efficient solutions (*bottom*). Note the high degree of concentration of pheromones on a path near the best path at bottom where the phase transition to collaborative foraging has occurred.

We note that this formulation of the arc transition probabilities uses the exponent α to lower the probabilities of choices with relatively low pheromones and emphasize those with relatively more. Note also that the τ_{min} and τ_{max} lower and upper bounds for arc-specific pheromones assure that these probabilities are never undefined, or converge to either one or zero.

Pheromone Update: The MIN-MAX pheromone update cycle is conditioned upon the best-so-far solution found by the ant population as a whole. The first step is to evaporate the pre-existing pheromones. Then the present solution is evaluated and replaces the previous best-so-far if it is better under *f(S)*. The arcs of the best solution are then updated with a specified amount of pheromone, possibly dependent on the merit of the solution. The *Min* threshold is applied so that no arc probability can go to zero and the Max threshold so no arc probability can approach one:

$$\forall i,j : arc(i,j).\tau \leftarrow (1-\rho)\cdot arc(i,j).\tau$$
$$f(S_t) < f(S_{best}) \Rightarrow S_{best} \leftarrow S_t$$
$$\forall arc(i,j) \in S_{best} : arc(i,j).\tau \leftarrow arc(i,j).\tau + ph(S_{best})$$
$$\forall arc(i,j) : arc(i,j).\tau \leftarrow \max(\tau_{\min}, arc(i,j).\tau)$$

Where $arc(i,j).\tau$ is the pheromone level associated with the arc, the evaporation rate is $0<\rho<1$; the minimum pheromone value is $\tau_{\min} > 0$, and S_t and S_{best} are solutions admissible under the constraint $\Omega(S)$. Importantly, this strategy decouples the pheromone update process from the individual ants; associating it instead with general evaporation and the best solution produced to time ***t*** by the entire community of foragers. Another point to note is that there is a finite upper bound on the pheromone at a given arc. Specifically, once a globally optimal solution has been found, then its arcs are updated by evaporation and best path award every turn and so it converges to the maximum pheromone amount τ_{max} given appropriate ρ.

The ACO parallel architecture and Method

The distributed ACO architecture is based on a model in which multiple copies of the problem graph and associated agent populations are distributed across many hosts and cores on *worker* processes, which periodically send results of local searches to a *master* node that combines them. Each worker node periodically sends the master node a copy of its local pheromone matrix and the worker nodes do not communicate directly with each other. The mechanism is iterated until the search has converged. The NDFS-II distributed middleware layer manages the worker node creation and dispatches them to the processors specified in the control center. The NDFS II simulation map and the relationships among the components are shown in Figure 3.

The master node server requests a list of all hosts willing to run the search from its peers. The user specifies the number of worker nodes per host. The master has each worker node run a new instance of an ACO agent-based simulator. The ACO simulation starts with the parameter distribution during which the master node creates a flow and sends the problem graph, evaporation rate, local ant population, and synchronization frequency to the worker nodes. The second phase of the simulation proceeds with the iterative data exchange. After a fixed number of local search iterations, the worker nodes enqueue the local pheromone matrices in the data flows to the master. The master node gathers this information, sums it and returns it to the worker nodes to update their local search spaces. Thus the distributed ant populations communicate through the stygmergic markings maintained by the master node, and collaborate effectively, even when each sub-population is below the coherence threshold.

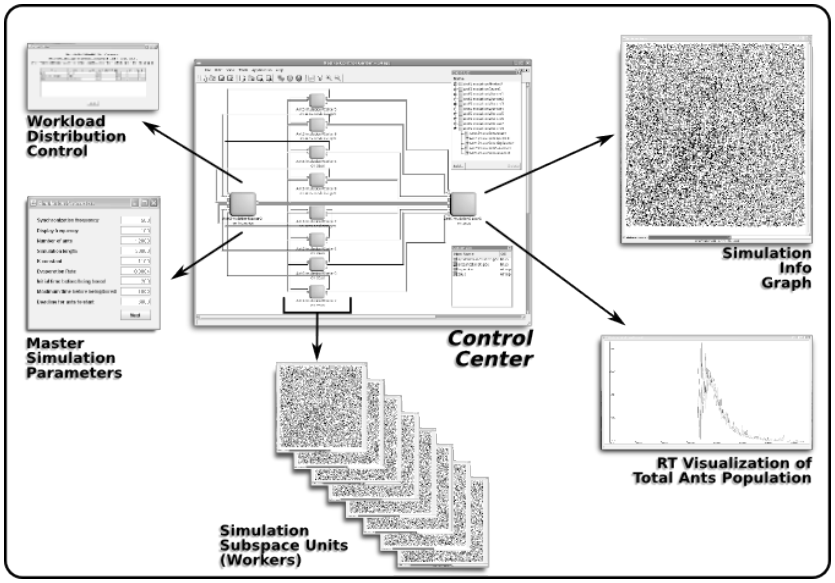

Figure 3. Control Center (*top center*) displays the NDFS-II application for distributed ACO, with a master node (*left in control center*), eight worker nodes (*center in control node*), and a display node (*right in control center*). Performance graphs (*bottom right*) show the solution length versus time, and the disposition of the global agent population (*top right*).

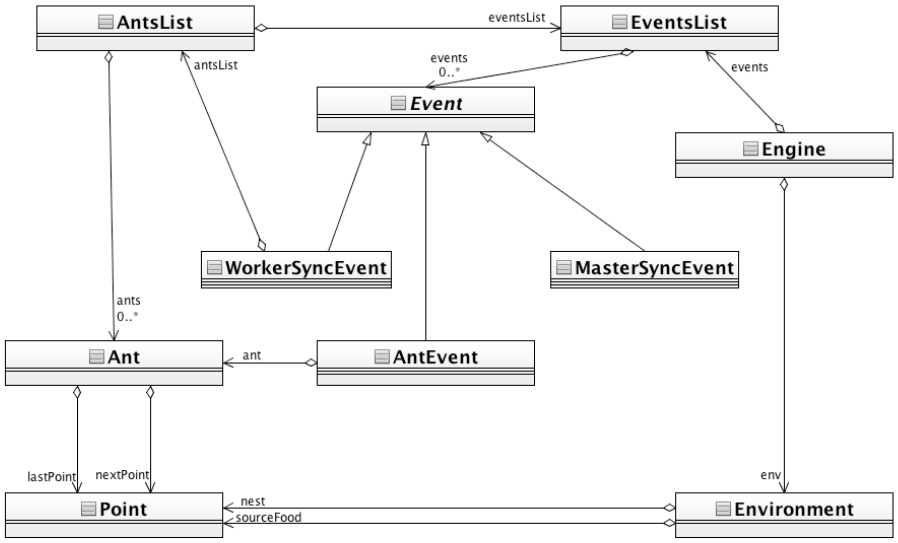

Figure 4. The major classes of the agent-based search and event queue include the *Engine, EventsList,* and *AntsList* classes with *Ant*, and *AntEvent* subclasses. These are used to implement a discrete event queue for ant agent movement through the graph, which the agents mark according to the desirability of the paths.

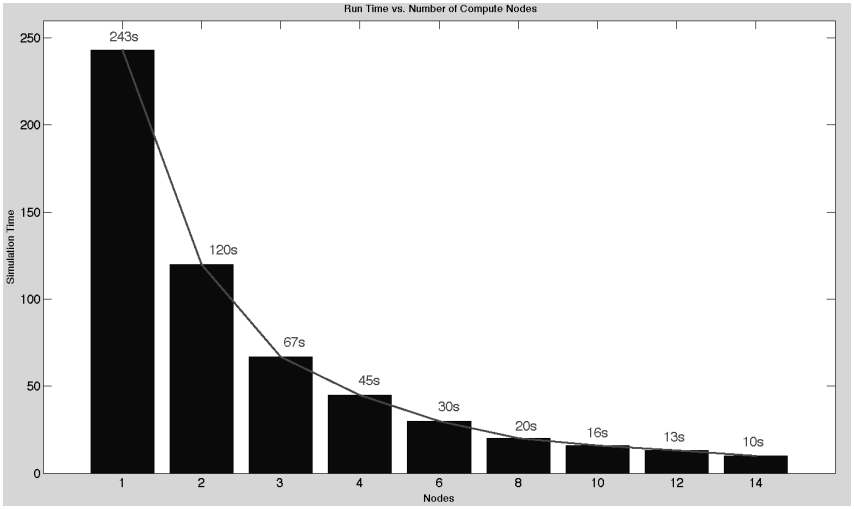

Figure 5. Scalability experiments with convergence times for one to fourteen compute cores decline throughout this range of parallelization. Note that as we go from eight worker nodes to 14, the convergence time is approximately halved.

The synchronization frequency can be in the range of hundreds of local agent cycles and still yield collaboratively obtained efficient solutions. Note, that depending on the problem size, the synchronization frequency must be chosen carefully in order to get an optimal solution in a reasonable time. Worker nodes can also extract the individual ant's positions for display node visualization. Master, worker and display nodes are independent Java programs and have a common JNI (Java Native Interface) library to wrap the NDFS C++ API.

Event agent-based Search and Synchronization

The agent-based search system is implemented as a discrete event queue in Java using the class structure shown in figure 4. The *Environment* class implements methods and data structures required to execute the search of a given problem space, then the ants search. The *Point* object represents a location in (x, y) coordinate space for the associated node. Each node of the problem graph has a successors attribute that specifies its adjacency list. The *Ant* class describes the behavior of an artificial ant by implementing methods such as "*choosePath"*, *"returnToNest"* or *"lookForFood"*. A forager *Ant* moves from one node to another in a time *T* determined by the length of the selected adjacent arc, and her speed. A discrete event queue is implemented in the simulation engine using the Bag Java objects as described by Luke et al. [17] rather than Vectors or Array lists, thus providing us direct access to the event list. The master-workers synchronization process is based on two events occuring every time-step. First the *"MasterSynEvent"* class

receives the pheromones matrices and creates an updated solution for the workers. Then the "*WorkerSynEvent"* class receives the new matrix from the master and applies its content to its search space.

Scalability of parallel ACO graph search algorithm

To quantify the parallel ACO algorithm performance under the NDFS-II framework we deployed it across multiple hosts with multiple compute cores. We used 5,000 ants in all of the searches, distributing them in equal numbers across the worker nodes. The search space consisted of a graph with ~10,000 nodes and ~ 19,000 arcs. We found that the solution quality depends on the master synchronization frequency between the distributed subpopulations and the time needed by the worker nodes to perform the search. Synchronization every one to two hundred ant queue events provides search performance that equals a single large population at reasonable communication overhead. Thus the result (the length of the best path found by the ants) can be as good as the search running on one processor but is obtained much faster. The execution time obtained under the NDFS-II experiments scales almost linearly for up to 14 worker nodes with a 24x speed increase, as shown in figure 5.

3. Conclusions

We introduced an enhanced NIST Data Flow System as a general parallel processing tool. Features that served its early deployment as a realtime sensor data-transport system provided a strong basis for more general parallel distributed processing capabilities. Research in Complex Systems agent-based simulations at large scales, and parallel computations in general were discussed. We described a parallel version of a combinatorial optimization algorithm based upon Ant Colony Optimization employing many simple ant agents with emergent behaviors with increasing agent populations. We showed very good scalability, suggesting that this particular system, and agent-based parallel algorithms in general have potential for application in very large graph search problems. Future work will include larger scale problems; variations on agent based systems; asynchronous rendezvous; and will study the use of local search methods to improve combinatorial optimization. All of our work is in the public domain and we are happy to collaborate with any interested researchers in parallel processing.

Disclaimer

Employees of the Federal Government developed the NDFS II in the course of their official duties. Pursuant to title 17 Section 105 of the United States Code this software is not subject to copyright protection and is in the public domain. Commercial products and open source projects are identified to adequately describe the subject matter of this work. This implies no endorsement. The NDFS is an experimental system and NIST assumes no responsibility for its use by other parties and makes no guarantees of its fitness for any particular purpose.

References

[1] A. Fillinger, I. Hamchi, S. Degre; L. Diduch, T. Rose, J. Fiscus and V. Stanford. Middleware and Metrology for the Pervasive Future. *IEEE Pervasive Computing Mobile and Ubiquitous Systems.* 8(3), 74-83, 2009.

[2] M. Dorigo and T. Stützle. *Ant Colony Optimization*. (MIT Press, Cambridge Massachusetts, 2004).

[3] M. Bolondi, and M. Bondaza: Parallelizzazione di un algoritmo per la risoluzione del problema del comesso viaggiatore; Master's thesis, Politecnico di Milano, 1993.

[4] E-G. Talbi, et al. Parallel ant colonies for the quadratic assignment problem. *Future Generation Computer Systems*, 17(4), 441-449, 2001

[5] R. Souto, et al. Reconstruction of Chlorophyll Concentration Profile in O*ff*shore Ocean Water using a Parallel Ant Colony Code. *Hybrid Metaheuristics*. 19–24, 2004.

[6] B. Bullnheimer, et al. Parallelization Strategies for the Ant System. *High Performance Algorithms and Software in Nonlinear Optimization*, Kluwer, Dordrecht, 87-100, 1998.

[7] P. Delisle, et al. Parallel implementation of an ant colony optimization metaheuristic with OpenMP. *Proc. of the 3rd European Workshop on OpenMP*, Barcelona, Spain, 2001.

[8] L. Diduch, et al. Synchronization Of Data Streams In Distributed Realtime Multimodal Signal Processing Environments Using Commodity Hardware. *Proc. of the IEEE International Conference on Multimedia & Expo (ICME)*, Hannover, Germany, June 2008.

[9] M. Michel, V. Stanford and O. Galibert. Network Transfer of Control Data: An Application of the NIST Smart Data Flow. *J. of Systemics, Cybernetics and Informatics*, 2, Jan. **2005**.

[10] V. Stanford, J. Garofolo, O. Galibert; M. Michel, C. Laprun. The NIST Smart Space and Meeting Room Projects: Signals, Acquisition, Annotation and Metrics. *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. 4 6-10 Hong Kong, April **2003**.

[11] P. Grassé. La reconstruction du nid et les coordinations inter-individuelles chez bellicostitermes natalensis et cubitermes. sp. la theorie de la stigmergie: essai d'interpretation du comportement des termites constructeurs. *Insectes Soc.* Vol 61 41-81. 1959.

[12] M. Beekman, D. Sumpter and F. Ratnieks. Phase transition between disordered and ordered foraging in Pharaoh's ants. ***PNAS***, Vol 98(17), p. 9703-9706, August 14, 2001.

[13] D. Sumpter, and S. Pratt. A modeling framework for understanding social insect foraging. *Behavioral Ecology Sociobiology* 53:131-144, 2003.

[14] M. Dorigo, Optimization, Learning and Natural Algorithms. *Ph.D. thesis Departimento di Elettonica, Politecnico di Milano*, Milan. 1992.

[15] Dorigo, Marco; E. Bonabeau, G. Theraulaz. Ant algorithms and stigmergy. *Future Generation Computer Systems*, 16 851-871, 2000

[16] T. Stützle and M. Dorigo. *A Short Convergence Proof for a Class of Ant Colony Optimization Algorithms.* IEEE Transactions on Evolutionary Computation, 6(4), August 2002.

[17] S. Luke, et al. *MASON: A New Multi-Agent Simulation Toolkit.* Proceedings of the 2004 SwarmFest Workshop.