Grid-based Asynchronous Replica Exchange

Zhen Li and Manish Parashar

Electrical and Computer Engineering Department
Rutgers University
94 Brett Road, Piscataway, NJ 08854
{zhljenny,parashar}@caip.rutgers.edu

***Abstract*— Replica exchange is a powerful sampling algorithm and can be effectively used for applications such as simulating the structure, function, folding, and dynamics of proteins and drug design. However, Grid-based implementations of the algorithm present significant challenges due to its synchronization and communication requirements. This paper presents an asynchronous formulation of the replica exchange algorithm and the design and implementation of a Grid-based asynchronous replica exchange engine (GARE). GARE is based on CometG, a decentralized computational infrastructure for Desktop Grid environments that provides a scalable communication and interaction substrate and presents a virtual semantically specialized shared space abstraction. It enables the dynamic and asynchronous interactions required by the algorithm to be simply expressed and efficiently implemented. The design and implementation of GARE/CometG and the replica exchange simulations that it enables are presented. Experimental evaluations using the PlanetLab [1] wide-area test bed as well as a campus Grid environment are presented.**

I. Introduction

Grid computing, based on the aggregation of large numbers of independent hardware, software and information resources spanning multiple organizations, is rapidly emerging as a dominant paradigm for distributed problem solving in a wide range of application domains. Complementary to Grid virtual organizations, Desktop Grids [2] leverage Internet connected computers to support large computations. Desktop Grid systems have been successfully used to address large applications in science and engineering with significant computational requirements, including global climate predication (Climatprediction.net) [3], protein structure prediction (Predictor@Home) [4], search for extraterrestrial intelligence (SETI@Home) [5], gravitational wave detection (Einstein@Home), and cosmic rays study (XtremWeb) [6]. While the successes of the above applications do demonstrate the potential of Desktop Grids, current implementations are typically limited to *embarrassingly parallel* [7] applications, where the individual tasks are independent and do not require inter-task communications. As a result, these implementations cannot support more general scientific and engineering applications where the tasks have communication and coordination requirements, such as applications based on replica exchange.

Replica exchange [8], [9], [10], [11] is an effective sampling algorithm proposed in various disciplines, such as bimolecular simulations where it allows for efficient crossing of high energy barriers that separate thermodynamically stable states. In this algorithm, several copies or replicas, of the system of interest are simulated in parallel at different temperatures using "walkers". These walkers occasionally swap temperatures based on a proposal probability, which allows them to bypass enthalpic barriers by moving to a higher temperature. The replica exchange algorithm has several advantages over formulations based on constant temperature, and has the potential for significantly impacting the fields of structural biology and drug design, such as the problems of structure based drug design and the study of the molecular basis of human diseases associated with protein misfolding.

Applications based on replica exchange tend to be computationally expensive and can benefit from the large numbers of processors and computational capacities offered by parallel and distributed computing systems and especially Desktop Grid environments. However, existing parallel implementations of replica exchange, specially in the structural biology community either target tightly coupled parallel systems such as the IBM BlueGene/L [12] or relatively small homogenous clusters. Further, these implementations are based on a simplified formulation of the algorithm that limits the potential power of the technique in two important aspects: (1) the only parameter exchanged between the replicas is the temperature of each replica, and (2) the exchanges occur in a centralized and totally synchronous manner, and only between replicas with adjacent temperatures. The former limits the effectiveness of the method and impedes temperature mixing, while the latter can limit scalability.

General formulations of the replica exchange algorithm require complex coordination and communication patterns between the walkers. Coupled with the complexity of the Grid environment, including its scale, its heterogeneity in computational, storage and communication capabilities, its dynamism and its unreliability, Grid-based replica exchange simulations present significant challenges. Another issue is the long running nature of replica exchange applications. These application can run for weeks and months, and as a result, resilience to failures is critical, specially in the case of cluster and grid-based implementations where mean time to failure can be smaller than the runtime of the application.

Clearly, the complexity of developing Grid-based replica exchange must be abstracted from the application scientists/engineers and effectively addressed by a computational infrastructure. Such an infrastructure should support dynamic walker management and efficient, robust and scalable exchanges to enable large scale simulations of the structure, func-

tion, folding, and dynamics of proteins. This paper presents the design, implementation and evaluation of such a computational infrastructure. It consists of two components: (1) an asynchronous formulation of replica exchange that is more suited to Grid environments and (2) a Grid-based asynchronous replica exchange engine (GARE). The asynchronous replica exchange formulation builds on our initial algorithm proposed in project Salsa [13], and extends it with following characteristics: (1) the exchanged parameters and the overall parameter ranges used by the simulation are determined at the beginning of the simulation and are known to all the walkers; (2) the parameters assigned to a walker only change when the walker performs an exchange; (3) exchanges can occur between walkers on different nodes; and (4) the walkers can dynamically join or leave the system. The first two observations allow individual walkers to locally determine the ranges of interest and enable exchange decisions to be made in a decentralized and decoupled manner. The third allows actual exchanges to occur between pairs of walkers in parallel. The last observation enables the replica exchange to deal with the environment and system dynamism.

The GARE builds on CometG [14] computational infrastructure for Desktop Grids, and extends it to support asynchronous replica exchange, including mechanisms for dynamic and anonymous task distribution, task coordination and execution, decoupled communication and data exchange. It provides a virtual shared space, by which the walkers can dynamically discover exchange partners, negotiate with them, and exchange data. Walkers periodically post temperature ranges that are of current interest for exchange to the space. If this range overlaps with the range of interest posted by another walker, an exchange can occur. The actual exchange is then negotiated and completed by the individual walkers in a peer-to-peer manner. As a result, exchanges are decoupled, dynamically and asynchronously determined. The GARE also allows walkers to dynamically join or leave the system, which can address the uncertainty of Desktop Grids and increase the robustness of simulations.

The rest of the paper is organized as follows. Section II describes the application domain and gives an overview of the replica exchange algorithm. Section III presents the design of CometG and GARE. Section IV describes the implementation and the experimental evaluation of Grid-based molecular dynamics simulations using GARE/CometG. Section V reviews related work. Section VI concludes the paper and outlines future research directions.

II. Parallel Replica Exchange for Structural Biology and Drug Design

The sequencing of the human genome, in conjunction with rapidly increasing efforts in structural genomics, is producing an explosion in the number of available high resolution protein structures. Molecular simulations of protein structural changes and drug binding to proteins depend critically on the design of highly efficient algorithms to search over the very rough energy landscapes which govern protein folding and binding. Scalable parallel replica exchange implementations can potentially address these molecular search problems and can significantly impact structure based drug design applications.

A. The Replica Exchange Algorithm

Replica exchange is an advanced canonical conformational sampling algorithm designed to help overcome the sampling problem encountered in biomolecular simulations. The method had been proposed independently on several occasions in various disciplines [8], [9], [10], [11]. In this method, several copies, or replicas, of the system of interest are simulated in parallel at different temperatures using walkers. These walkers occasionally swap temperatures based on a proposal probability that maintains detailed balance [15]. Note that general formulations of replica exchange simulations allow walkers to exchange multiple parameters, e.g., temperature plus energy. However current implementations only exchange temperatures.

These exchanges allow individual replicas to bypass enthalpic barriers by moving to high temperatures. A parallel version of this algorithm was proposed by Hukushima and Nemoto [11]. The replica exchange algorithm is easy to implement and does not require time-consuming preparatory procedures. Further, it can decrease the sampling time by factors of 20 or more, as compared to constant temperature molecular dynamics when applied to peptides at room temperature [16]. Details of the algorithm can be found in [15] and application examples can be found in [17], [18].

The molecular dynamics replica exchange canonical sampling method has been implemented in the IMPACT (Integrated Modeling Program, Applied Chemical Theory) molecular mechanics program [19], and is the molecular simulation method used in this research. The implementation follows the approach proposed by Sugita and Okamoto [17]. The method consists of running a series of simulations at fixed specified temperatures. Each replica corresponds to a temperature. An exchange of temperatures between replicas i and j at temperatures T_m and T_n is attempted periodically and is accepted according to the following Metropolis transition probability [17]:

$$W = \min\{1, \exp[-(\beta_m - \beta_n)(E_j - E_i)]\} \qquad (1)$$

where $\beta = 1/kT$ and E_i and E_j are the potential energies of replicas i and j, respectively. After a successful exchange, the velocities of replicas i and j are rescaled at the new temperature.

B. Existing Parallel Implementations of Replica Exchange-based Molecular Dynamics Simulations

Molecular dynamics programs are essentially loops over a large number of integration steps, each of which advances the time forward for one step. Replica exchange is attempted periodically after a chosen interval of steps. As mentioned in the introduction, existing parallel implementations of replica exchange are MPI [20]-based, centralized and synchronous, and target relative small tightly coupled homogenous systems. For example, in the existing implementation in IMPACT, a central

master node collects temperature data about all the replicas from the walker nodes, and then broadcasts the collected data array to the walkers. Each walker node receives this data array and sorts the array locally. Neighboring temperatures in the sorted array are potential partners for temperature exchange. The master node randomly selects between two modes of exchange. One is to exchange with upper neighboring temperature and the other is to exchange with lower neighboring temperature. The master notifies the walkers about the selected mode, and walkers can then mutually exchange temperatures based on this information. During the actual exchange, one of the two walker nodes with neighboring temperatures in the sorted array that are paired up for temperature exchange, acts as a temporary server. This walker collects temperature and potential energy data from the other node, determines whether the exchange is feasible based on the transition probability given in Eq. (1), and replies with either the new temperature, if the exchange is successful, or with a notice of denial otherwise.

The parallel replica exchange implementation described above has several limitations. First, the scheme limits the exchange to only neighboring temperatures. This limitation is not a concern when the number of replicas is small and there is a small chance of exchange between non-nearest temperatures. However, as the number of processors (and correspondingly walkers) increases, the difference between target temperatures becomes small enough to allow exchanges between non-nearest neighbor replicas. In such cases, more flexible schemes which allows non-nearest neighbor temperature exchange are desirable. Second, the implementation is based on a centralized master that gathers and scatters data system wide. Gathering data from all the nodes on a single node may be infeasible in large systems, and a centralized master can quickly become a bottleneck. Further, gather and scatter operations are synchronous and expensive. Also, since the master node also participates in the simulation as a walker, there is a load imbalance which can lead to additional synchronization overheads.

To address the above limitations, we proposed an initial asynchronous realization of the replica exchange algorithm in project Salsa [13]. This formulation distinguished itself from existing implementations in two aspects: (1) it allows arbitrary walkers with mutual interesting range to exchange temperature; (2) it enables the temperature exchanges in an asynchronous and parallel manner, and is used in this project. However, Salsa still targeted closely coupled and reliable cluster environments and only supported a single walker per node. This research targets Grid environments, and builds on the initial Salsa asynchronous replica exchange formulation to address platform heterogeneity, environment unreliability, and dynamic walker management.

III. A Grid-based Asynchronous Replica Exchange Engine

The Grid-based asynchronous replica exchange engine (GARE) builds on CometG [14], a decentralized computational infrastructure for Desktop Grid environments, and extends it to support asynchronous replica exchange.

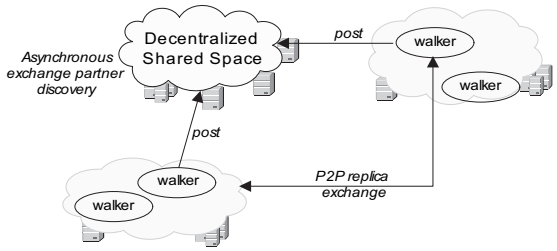

Fig. 1. A conceptual overview of the GARE/CometG infrastructure.

A. GARE/CometG System Architecture

GARE/CometG provides abstractions and mechanisms to support scalable parallel implementations of the general replica exchange formulation, where walkers can exchange non-nearest neighbor temperatures in a decoupled, decentralized, and asynchronous manner. Figure 1 presents a conceptual overview of the GARE/CometG infrastructure. It provides a virtual decentralized shared space abstraction that can be associatively accessed by all walkers. Walkers can use this space to dynamically discover exchange partners and negotiate with them, and exchange data. Walkers periodically post temperature ranges that are of current interest to the space. If this range overlaps with the range of interest posted by another walker, an exchange can occur. Then the individual walkers negotiate and complete the actual data exchange in a peer-to-peer manner. As a result, exchanges are decoupled, dynamically and asynchronously determined, and not limited to neighboring temperatures. The system has a layered architecture (shown in Figure 2) consists of 3 main layers described below.

a) Communication Layer: The communication layer provides an associative communication service and guarantees that content-based messages, specified using flexible content descriptors, are served with bounded cost. This layer also provides a direct communication channel to efficiently support data transfers between peer nodes. The communication channel is implemented using a thread pool mechanism and TCP/IP sockets.

This layer has 2 key components - a content-based routing engine and a one-dimensional structured self-organizing overlay. The routing engine implements a distributed hash table (DHT) and maps messages based on their content descriptors to peer nodes. The message content descriptors are keyword/value pairs and are derived from a domain-specific k-dimensional (kD, $k \geq 1$) information space. The mapping is based on the functions that map this kD space to a linear node index space, which is then dynamically partitioned across the participating peer nodes. Each node is responsible for its portion of the index and the corresponding region of the kD space, and essentially serves as the rendezvous point for messages intersecting with this region.

The current routing engine uses the Hilbert SFC [21] to map the information space to the linear node index space. The Hilbert SFC is a locality preserving continuous and recursive mapping from a kD space to a 1D space. The routing engine supports flexible querying using partial keywords, wildcards, or ranges and readily extends to any number of dimensions. Its locality preserving and recursive nature, i.e. points that are close on the curve are mapped from points that are close in the kD space, enables the index space to maintain content locality and efficiently resolve content-based lookups [22]. Note that in the case of replica exchange implementations that only use temperature exchange, i.e., k = 1, CometG uses simple hashing where the index is directly derived from the overall temperature range used by the simulation. The routing engine provides a single operator, *put(keys, data)* where *keys* are derived from the application information space and *data* is the message payload.

The overlay network is composed of peer nodes, which may be any node in the Desktop Grid system (e.g., end-user computers, servers, or message relay nodes). The peer nodes can join or leave the network at any time. The CometG architecture is not tied to any specific overlay topology. In the current implementation, we use Chord [23] which has a ring topology, primarily due to its guaranteed performance, efficient adaptation for node joining and leaving, and the implementation simplicity. In principle, this overlay could be replaced by other structured overlays. The overlay provides the *lookup(identifier)* operator. Given an identifier, this operation locates the node that is responsible for it, i.e., the node with an identifier that is the closest identifier greater than or equal to the queried identifier.

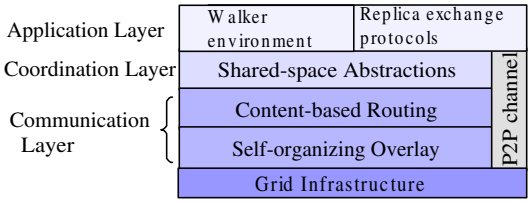

Fig. 2. An schematic of the GARE/CometG stack.

b) Coordination Layer: The coordination layer enables discovery of potential exchange partners between walkers. This layer supports the tuple space abstractions [24], including *Out*, *In*, and *Rd* operators. *Out(ts, t)* is a non-blocking operation that inserts tuple t into space ts. *In(ts, $\bar{t}$, timeout)* is a blocking operation that removes a tuple t matching template $\bar{t}$ from the space ts and returns it. If no matching tuple is found, the calling process blocks until a matching tuple is inserted or the specified *timeout* expires. In the latter case, $null$ is returned. *Rd(ts, $\bar{t}$, timeout)* performs exactly like the *In* operation except that the tuple is not removed from the space.

The primary components of the coordination layer are a data repository for storing tuples and templates, a local matching engine, and a message dispatcher that interfaces with the communication layer to translate the coordination primitives to content-based routing operations at communication layer and vice versa. The tuples in CometG are represented as simple XML strings as they provide small-sized flexible formats that are suitable for efficient information exchange in distributed heterogeneous environments.

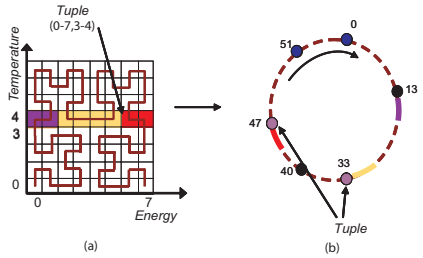

Fig. 3. Example of tuple post in 2D space: (a) the tuple defines a rectangular region in the 2D space consisting of 3 clusters; (b) the region is mapped to the overlay index and the tuples are routed to the node 33 and 47.

The coordination primitives are implemented using the content-based messaging abstraction provided by the communication layer. Using the keywords associated with a tuple, the tuple is routed to the appropriate peer nodes in the overlay using following steps. (1) Keywords are extracted from the tuple and used to generate the *keys* for the *put* operation. The payload of the message includes the tuple data and the coordination operation. (2) The routing engine uses the DHT mapping function to identify the indices corresponding to the *keys* and the corresponding peer nodes. (3) The overlay *lookup* operation is used to route the tuple to the appropriate peer nodes.

c) Application Layer: The application layer provides an environment for dynamically managing walkers and protocols for asynchronous exchange. Walkers are extensions of CometG computational tasks and the walker environment configures, initiates, monitors, and manages the local walkers. Walkers are dynamically dispatched to CometG nodes during node initiation. The protocol provides three operators for implementing the asynchronous replica exchange algorithm:

- *post (t)*: inserts a tuple t into space. This operator is used by a walker to express its desire to exchange. Tuple t contains the details of the exchange including a specification of the parameters and ranges (upper and lower bounds) of interest.
- *query (walkerid, timeout)*: sends a query message to a potential partner. The calling walker blocks until receiving a "confirm" or "refuse" response or the specified *timeout* period expires.
- *getp (walkerid, d, timeout)*: exchanges a walker's data d with a selected partner. If the attempt to exchange is successful, the calling walker blocks until the exchange is finished or the specified *timeout* expires. If the attempt fails, *getp* returns with a failure code.

The *post* operator is implemented using the *Out* operation provided by the coordination layer. Figure 3 illustrates the *post* operation in a 2D replica exchange simulation. In this example,

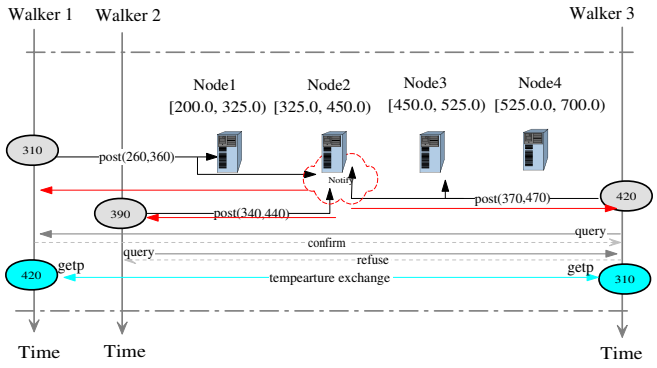

Fig. 4. Operation of a sample asynchronous replica exchange.

the tuple specifies ranges of interest for the two parameters, which define a rectangular region in the information space. The Hilbert SFC is used to map this region to appropriate index spans in the linear index space and the corresponding peer nodes to which these index spans have been mapped. The tuple is then routed on the overlay to these nodes. To guarantee data insertion, each *post* request is confirmed by responses from the corresponding destination peer nodes. The *getp* and *query* operators are implemented using the peer-to-peer communication channel to ensure efficient data exchange.

B. Grid-based Asynchronous Replica Exchange

The operation of asynchronous replica exchange using GARE/CometG is illustrated in this section using a temperature exchange example. The operation consists of three phases: (1) the *post* phase in which, candidate exchange partners are identified and notified; (2) the *query* phase in which, potential exchange partners negotiate and agree to exchange; and (3) the *getp* phase in which, confirmed partners attempt to exchange data. When a walker attempts to exchange its current temperature, it computes the target temperature range that it is willing to exchange with, and posts this range using the *post* operator. Based on the temperature range posted, the request is routed to all the nodes whose index ranges overlap with the posted range. When a remote *post* request is received by a peer node, it first checks its local repository for potential exchange partners that have previous posted interests with overlapping temperature ranges. If one or more potential exchange partners are found, the corresponding walkers are notified. Otherwise, the incoming request is stored.

The process is illustrated in Figure 4. In this figure, the ranges of interest of $walker_1$, $walker_2$, and $walker_3$ overlap. When the relevant node receives the post from $walker_3$, it discovers that $walker_1$ is a potential exchange partner and notifies the two walkers. Then, $walker_3$ queries $walker_1$ to see whether it is available for an exchange. This is necessary because, even though $walker_1$ has expressed a desire to exchange, it may have already partnered with another walker or may have decided to give up and continue with its computations. On receiving this query, $walker_1$ checks its local state, which can be either "free" or "pending". A walker is available for an exchange only if it is in the "free" state. The "pending" state indicates that the walker is either exchanging with another walker or has committed to exchange with another walker but the exchange has not yet occurred. Since $walker_1$ is in the "free" state, it responds affirmatively to $walker_3$ and commits to the exchange. Once both walkers confirm their intents to exchange, they change their states to "pending" and perform data exchange using the *getp* operator. Since $walker_3$'s post also matches $walker_2$'s interest, $walker_3$ receives a query from $walker_2$. Since $walker_3$ has already committed to exchange with $walker_1$ and is in the "pending" state, it refuses this request. In this example, $walker_2$, rather than attempting an exchange with another potential partner, continues computing using its current data and waits until the next exchange cycle to attempt an exchange.

Once a pair of walkers agree to exchange, they initiate the actual exchange by invoking the *getp* operator, which proceeds as follows. One of the walkers sends its current data (e.g. temperature and energy) to its potential partner. The potential partner determines whether the exchange can be completed based on the data it receives and its own data. This step is necessary since the exchange happens asynchronously and in parallel with the computation, and a walker's data (i.e., energy) may have changed since it posted its exchange interest. If the walker decides to continue with the exchange, it will send an exchange acceptance to its partner along with its current local data. It will then wait for a similar acceptance from the partner to complete the exchange. Note that an exchange is between a pair of walkers and multiple exchanges between different pairs of walkers can proceed in parallel. After the exchange is completed, both walkers remove posted tuples from the space since these tuples and the data they contain are no longer valid.

Since a *post* request typically maps to multiple peer nodes and each node may find more than one partner, it is possible that a requesting walker is notified of multiple candidates located on different nodes. In this case, the first notification that reaches the requesting walker is accepted. In the algorithm, a walker specifies the ranges for each parameter that it is interested in exchanging as part of the *post* operator. Usually, the larger the range is, the higher is the probability of finding an exchange partner and results in better solution quality. However, a larger range will also map to a large number of nodes, which in turn increases communication overheads as well as the load at the nodes, and reduces system performance. In the current system, the *post* operator randomly selects a subset of the nodes to which the interval is mapped, and forwards the post request to these nodes. The size of this subset can be configured by users to achieve desired tradeoffs between solution quality and simulation performance.

If the parameter ranges are not evenly distributed, the posted ranges will result in load balancing issues. In the current implementation, the fact that the parameter ranges are known is used to define a simple load balancing protocol. The distribution of parameter ranges within the linear index space can be analyzed and this analysis can be used while partitioning the index space across the nodes to ensure that the

system is load-balanced. Since more general replica exchange formulations may use dynamically defined ranges, we are working on a dynamic load-balancing protocol.

C. Addressing Grid Dynamism and Unreliability

The GARE/CometG provides fault tolerance mechanisms to address the dynamism and unreliability inherent in Grid environments. These mechanisms assume a fail-stop failure model and timed communication behavior [25]. Under these assumptions, possible failures include walker failure, posted data loss, and negotiation failure. These failures are addressed as follows:

- *Walker failures* are handled using checkpoint-restart. The walker environment periodically checkpoints the local state of each walker, such as its current exchange parameters, to a stable storage. When a walker fails, it can be restarted using this checkpoint. Currently, the detection of walker failures and walker restarts are manual. Note that due to the asynchronous nature of the algorithm, other walkers are not affected by the failure and restart of a walker except that any attempt to exchange with this walker will not succeed.
- *Loss of posted data* occurs only when the node at which a tuple is stored fails since tuple insertions are guaranteed. From a walker's point of view, the impact of this failure is that its attempt to exchange will not succeed and it will repost its request tuple in the next exchange cycle. The resilience of the overlay (e.g., Chord's ability to route around failures [23]) guarantees that the repost will be routed to an operation peer node.
- *Negotiation failures* may result due to the failure of a walker, loss of a message, failure of a communication link, or failure or departure of a node. These failures are handled using timeouts for the *query* and *getp* operations. Once again, due to the resilient nature of asynchronous algorithms, the application is not affected.

Further, redundancy in storage and routing can be incorporated within the overlay as described in [26], where a group of nodes act as one peer. In this case, the consistency of the tuple space as well as issues of group synchronization, such as degree of redundancy, group membership, group communication protocol, etc., must be addressed.

IV. Implementation and Experimental Evaluation

The current prototype of CometG and the Grid-based asynchronous replica-exchange engine (GARE) has been implemented on Project JXTA [27], a platform independent peer-to-peer framework. The JXTA platform provides a virtual network for applications, which can cross barriers such as firewalls/NATs to establish peer communities spanning any part of the physical network. JXTA peers can discover peer resources, communicate with each other, and self-organize into peergroups. A JXTA peergroup provides a scoping mechanism, using which messages are only propagated among group members. JXTA also provides security features that can be used by applications. Each CometG node operates as a JXTA peer identified by a *JxtaID*. Each node can support multiple walkers. Nodes in CometG organize using *JXTA Discovery Protocol* to form the overlay. The communication layer of CometG maps the logical overlay peer identifer to the node's *JxtaID*, and uses the *JXTA Resolver Protocol* for communication.

The overall operation of CometG consists of *bootstrap* and *running* phase. The *bootstrap* phase is used to setup the overlay. During this phase, peer nodes join the CometG JXTA peergroup and exchange messages with the rest of the group. In the *running* phase, the system runs both in *stabilization* and *user* modes. In the *stabilization* mode peer nodes manage the structure of the overlay. In this mode, peer nodes respond to periodic queries from other peers to ensure that routing tables are up-to-date and to verify that other peer nodes in the group have not failed or left the system. In the *user* mode, peer nodes participate in user applications. In this mode, application developers can configure the system, setup application specific parameters and initiate the application.

A. Experimental evaluation

GARE/CometG has been deployed on a wide-area environment using PlanetLab [1] test bed and a small heterogeneous Grid at Rutgers. The objectives of the experiments presented in this section are to demonstrate the ability of GARE/CometG to support wide-area deployments of replica exchange applications and to evaluate performance and scalability.

A temperature replica exchange simulation, which is based on the IMPACT framework, is used in the experiments presented below. In this evaluation, all the experiments are configured to run for 10,000 sampling cycles, and exchange is attempted every 25 cycles. Temperatures are distributed eventually within the 200-700 K range. At equilibrium each walker should visit each temperature with equal probability. The rate of temperature equilibration is measured by the number of "cross-walks", where a walker originally within the low temperature range (200 K $\leq$ T $\leq$ 250 K) reaches the upper temperature range (650 K $\leq$ T $\leq$ 700 K) and then returns to the lower temperature range. The number of "cross-walks" is measured in the experiments to evaluate the system performance - the larger the number of crosswalks for a run, the better is the performance of the simulation. In the experiments presented below, the temperature range posted by walkers was set to a window of size 400K and 200K around its target temperature, i.e., [temp - 200K, temp + 200 K] and [temp - 100K, temp + 100 K].

The first set of experiments were conducted on a Grid consisting of heterogeneous Linux-based computers on the Rutgers campus network. Each computer runs a single CometG instance and supports 4 walkers. Table I shows the number of temperature cross-walks measured for decentralized replica exchange simulations with 8, 16, 32, 64, and 128 walkers, compared with the corresponding number of cross-walks obtained using a traditional centralized approach where exchanges are all conducted at a central (master) node. The latter case was achieved by mapping the CometG shared

TABLE I

NUMBER OF TEMPERATURE CROSS-WALK EVENTS.

Number of walkers	8	16	32	64	128
Posted temperature range	decentralized simulation				
$[-200K, 200K]$	49	91	115	251	582
$[-100K, 100K]$	11	43	82	113	262
Posted temperature range	centralized simulation				
$[-200K, 200K]$	74	119	150	178	202
$[-100K, 100K]$	26	55	81	92	126

space to a single peer node. As shown in the table, the number of observed temperature cross-walks increases with increasing numbers of walkers and the posted temperature range. The decentralized implementation achieves more cross-walks than the centralized approach when the number of walkers is greater than 64, although the centralized approach achieves more crosswalks for a small number of walkers. This is because a centralized node quickly becomes a bottleneck as the number of walkers increases. The average wall-clock execution time of the simulation for different numbers of walkers are plotted in Figure 5. As seen in the figure, the decentralized implementation scales well, while as expected, the centralized implementation does not scale. The impact of centralization is even more pronounced for larger systems in wide-area Grid environments.

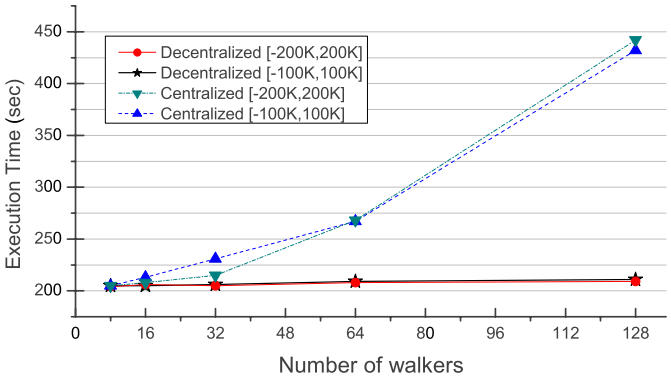

Fig. 5. Average wall-clock execution time for different numbers of walkers.

The second set of experiments measured the number of temperature cross-walks for larger numbers of walkers in the decentralized implementation. The GARE/CometG-based replica exchange implementation supports non-nearest neighbor temperature exchanges, which is essential for ensuring proper mixing of temperatures across the walkers, especially when the number of walkers is large. This experiment used a fixed system size of 32 nodes and evenly distributed the walkers across these nodes. Figure 6 plots the number of cross-walks for temperature range [-200K, 200 K] and [-100 K, 100 K] using 512, 640, 960, and 1280 walkers. These results illustrate the effects of increasing the temperature range on the number of cross-walks. The results also demonstrate the ability of GARE/CometG to effectively support the increased communication due to larger numbers of walkers and larger temperature ranges.

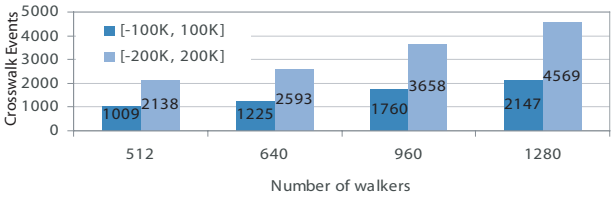

Fig. 6. Effect of the posted temperature ranges on the number of temperature crosswalk events for large numbers of walkers.

The third experiment was conducted on the wide-area PlanetLab [1] test bed. PlanetLab is a large scale heterogeneous distributed environment composed of inter-connected sites on a global scale. The goal of this experiment is to demonstrate the ability of GARE/CometG to support replica exchange applications in unreliable and highly dynamic environments such as PlanetLab, which essentially represents an extreme case for a Desktop Grid environment. GARE/CometG was deployed more than 200 machines on PlanetLab, however, only a fraction of these nodes could be effectively used at anytime. This experiment was conducted on January 03, 2007. In the experiment, we used temperature range at [-200K, 200K] and 32 walkers, which were dynamically mapped to nodes that joined the replica exchange space. Walkers were dynamically initialized on the nodes (up to 4 walkers per node). The timeout threshhold of *getp* and *query* operation was set to 5 seconds. These walkers dynamically joined the application, started their computation and performed exchanges, and left

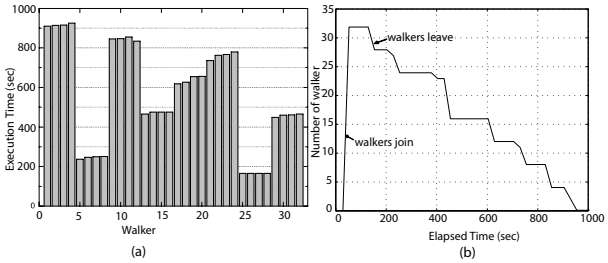

Fig. 7. Evaluation of GARE/CometG-based replica exchange on the PlanetLab wide-area test bed. (a) Execution time of the different walkers. (b) Change in the number of walkers due to the dynamism of the environment.

the system when their computation terminated or the node failed or lost connectivity. The joining or leaving of walkers did not impact the execution of other walkers. The number of walkers was monitored during the experiment. The application terminated after all the walker finished their sampling cycles, in which 86 cross-walk events were observed. The results of the experiment are plotted in Figure 7. Figure 7 (a) plots the execution time for each walker. This plot illustrates the heterogeneity of the PlanetLab nodes. Figure 7 (b) demonstrates the fluctuations in the number of walkers during the lifetime of the application due to the dynamism of the environment.

V. Related Work

Existing parallel/distribute replica exchange implementations are based on a simplified formulation of the replica exchange algorithm as described previously, and use a centralized master to periodically schedule and manage exchanges. These implementations either directly build on sockets, as in [15], or use message passing libraries such as MPI or PVM, as in [28]. In [28], a parallel replica exchange implementation is developed specially for a ring topology, which is suitable for systems where the processors can be configured as a ring and supports blocking send and receive calls. Like the MPI-based implementations, this implementation also supports only nearest neighbor temperature exchanges.

The Folding@home [29] project proposed a multiplexed replica exchange algorithm, which uses multiplexed-replicas with a number of independent molecular dynamics runs at each temperature, and attempts exchanges of configurations between these multiplexed-replicas. In this formulation, the efficiency of the simulation is enhanced as a number of independent molecular dynamics simulation replicas are run at each temperature and there are a larger number of potential exchange partners available. Further, the multiplexing between replicas is arranged in such a way that the discrepancy between exchange partners is reduced. In contrast, the asynchronous replica exchange approach presented in this paper improves simulation efficiency by eliminating the limitation of nearest neighbor exchanges, instead of introducing redundant computations. This not only improves scalability but also improves efficiency by enabling non-nearest neighbor temperature exchanges, which is desirable for simulations with a large number of replicas. To the best of our knowledge, this work is the first to address the decentralized asynchronous parallel implementation of replica exchange in Grid.

VI. Conclusion and Future Work

This paper focused on enabling large-scale Grid-based replica exchange simulations. The paper presented an asynchronous formulation of the replica exchange algorithm that enables non-nearest neighbor exchanges in a decentralized and asynchronous manner. The formulation improves both, the effectiveness and efficiency of the algorithm and is suitable for Grid-based implementations. The paper also described the design and implementation of a Grid-based asynchronous replica exchange engine, which enables replica exchange simulations on Desktop Grid environments. The feasibility and performance of the engine was experimentally evaluated using Rutgers campus networks and the PlanetLab wide-area test bed.

While the presented system can effectively support thousands of walkers, its scalability can be further improved to even millions of walkers using two enhancements: (1) separating the space nodes from end nodes, where the space nodes provide coordination services and the end nodes host the walkers; (2) employing relatively powerful peers, i.e., super-peers, with larger memory capacity and network bandwidth, as space nodes. This enhancements are currently being explored.

Acknowledgment

The research presented in this paper is supported in part by National Science Foundation via grants numbers CNS 0305495, CNS 0426354, IIS 0430826 and ANI 0335244, and by Department of Energy via the grant number DE-FG02-06ER54857. The authors would like to express thanks to Li Zhang, Emilio Gallicchio, and Ronald M. Levy for their contributions during this research.

References

[1] "Project PlanetLab," http://www.planet-lab.org.
[2] D. Kondo, M. Taufer, C. Brooks, H. Casanova, and A. Chien, "Characterizing and evaluating desktop grids: An empirical study," in *Proceedings of the IPDPS*, 2004.
[3] "Climatprediction.net," http://climateapps2.oucs.ox.ac.uk/cpdnboinc/.
[4] "Predictor@home," http://predictor.scripps.edu/.
[5] "Seti@home," http://setiathome.ssl.berkeley.edu/.
[6] "Xtremweb," http://xw.lri.fr:4330/XtremWeb/.
[7] B. Wilkinson and M. Allen, *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. Prentice Hall, 2004.
[8] R. Swendsen and J. Wang, "Phys. rev. lett." vol. 57, 1986.
[9] C. Geyer and E. Thompson, "J. am. stat. assoc."
[10] E. Marinari and G. Parisi, "Europhys. lett." vol. 19, 1992.
[11] K. Hukushima and K. Nemoto, "J. phys. soc. jpn." vol. 65, 1996.
[12] M. E. et al., "Parallel implementation of the replica exchange molecular dynamics algorithm on blue gene/l," in *Proceedings of Parallel and Distributed Processing Symposium*, 2006.
[13] R. L. L. Zhang, M. Parashar and E. Gallichio, "Salsa: Scalable asynchronous replica exchange for parallel molecular dynamics applications," in *Proceedings of The 2006 International Conference on Parallel Processing*, 2006, pp. 127–134.
[14] Z. Li and M. Parashar, "A decentralized computational infrastructure for grid based parallel asynchronous iterative applications," *Journal of Grid Computing*, vol. 4, no. 4, December 2006.
[15] S. G. H. Nymeyer and A. E. Garcia, "Atomic simulations of protein folding using the replica exchange algorithm," vol. 383, pp. 119–149, 2004.
[16] K. Y. Sanbonmatsu and A. E. Garcia, "Proteins," vol. 46, 2002.
[17] Y. Sugita and Y. Okamoto, "Replica-exchange molecular dynamics method for protein folding," vol. 314, pp. 141–151, 1999.
[18] E. G. A. K. Felts, Y. Harano and R. M. Levy, "Free energy surfaces of β -hairpin and α -helical peptides generated by replica exchange molecuar dynamics with agbnp implicit solvent model," vol. 56, pp. 310–321, 2004.
[19] J. B. et al., "Integrated modeling program, applied chemical theory (impact)," vol. 26, pp. 1752–1780, 2005.
[20] "The message passing interface (mpi) standard," http://www-unix.mcs.anl.gov/mpi/.
[21] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz, "Analysis of the clustering properties of the hilbert space-filling curve," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 1, pp. 124–141, 2001.
[22] C. Schmidt and M. Parashar, "Enabling flexible queries with guarantees in p2p systems," *IEEE Network Computing, Special issue on Information Dissemination on the Web*, no. 3, pp. 19–26, June 2004.
[23] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable Peer-To-Peer lookup service for internet applications," in *Proceedings of the SIGCOMM*, 2001, pp. 149–160.
[24] N. Carriero and D. Gelernter, "Linda in context," *Communications of the ACM*, vol. 32, no. 4, pp. 444–459, Apr. 1989.
[25] F. Cristian and C. Fetzer, "The timed asynchronous distributed system model," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 6, pp. 642–657, 1999.
[26] C. Schmidt, "Flexible information discovery with guaratees in decentralized distributed systems," Ph.D. dissertation, Rutgers University, 2005.
[27] "Project JXTA," Internet: http://www.jxta.org.
[28] D. v. d. S. H. J. C. Berendsen and R. van Drunen, "Gromacs: a message-passing parallel molecular dynamics implementation," *Computer Physics Communications*, vol. 91, pp. 43–56, 1995.
[29] "Folding@home," http://folding.stanford.edu/.