Efficient Large-Scale Replica-Exchange Simulations on Production Infrastructure

By Abhinav Thota[1,2], André Luckow[1] and Shantenu Jha[1,2,3]
[1]*Center for Computation & Technology, Louisiana State University, Baton Rouge, LA 70803, USA*
[2]*Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803, USA*
[3]*e-Science Institute, Edinburgh EH8 9AA, UK*

Replica-Exchange (RE) Methods which represent a class of algorithms that involve a large number of loosely-coupled ensembles and are used to understand physical phenomena – ranging from protein folding dynamics to binding affinity calculations. We develop a framework for RE that supports different replica pairing and coordination mechanisms, that can utilise a range of production cyberinfrastructure concurrently. We use this framework to implement three different formulations of the RE algorithm. We characterise the performance of the different RE algorithms at unprecedented scales on production distributed infrastructure.

Keywords: Replica-Exchange, SAGA, Large-Scale, Production

1. Introduction

The Replica-Exchange (RE) (Hansmann 1997, Sugita and Okamoto 1999) methods – which represent a class of algorithms that involve a large number of loosely-coupled ensembles. RE simulations are used to understand physical phenomena – ranging from protein folding dynamics to binding affinity calculations. The design and development of most RE implementations (Woods et al. 2005) is influenced and constrained by the programming systems and the infrastructure it is developed against. Breaking this coupling between the development and the underlying infrastructure, to enable applications to be flexible (across infrastructure), extensible (to new methods of communication and coordination) and scalable is an important design objective of distributed applications – both logically distributed and physically distributed.

Application formulations that are scalable while being flexible and extensible are better suited to using the diverse range of traditional and hybrid infrastructure (e.g., grid-cloud and heterogeneous resources). Along with application formulations that facilitate the flexible utilisation of a range of infrastructure, it is imperative to have the correct runtime abstractions that support flexible deployment of these applications. In support of flexible and scalable formulations of the RE class of algorithms, we develop a RE Framework that supports multiple formulations, is extensible to a broad range of infrastructure and as we shall show scales-up and scales-out. Our RE Framework utilises a flexible pilot-job implementation (SAGA BigJob) to support the execution of the ensembles. It supports scalable implementation of RE that can utilize a range of infrastructure concurrently that supports different coordination mechanisms different exchange coordination mechanisms (synchronous versus asynchronous), and thereby different variants of the RE algorithm.

The paper is organised as follows. § 2 sketches out the three different RE algorithms that are investigated; we also present an approximate mathematical model for the different algorithms. § 3 outlines the architecture of the RE Framework – the SAGA BigJob (how it supports the dynamic execution of multiple replicas) and other important elements that make the framework flexible and extensible. In § 4, we present our implementation of the RE algorithms and understand the primary determinants of performance and relate it to the mathematical model of § 2. In § 5, we describe the experiments performed to assess and understand performance when scaling-up (on a single machine) as the number of replicas increases. We compare and analyze the performance of the different RE formulations (synchronous and asynchronous) when scaled-up to 256 replicas as well as when scaled-out to use more than one machine. § 6 concludes the paper and discusses future work.

2. Replica-Exchange Algorithms

The RE class of algorithms involves the concurrent execution of *replicas* - which are defined as instances of essentially similar simulations but with minor differences, such as the defining temperature of the replica. These replicas are loosely-coupled, in that there is an infrequent exchange between pairs of replicas; in addition to the frequency of communication between the replicas being low (relative to within a single replica), the amount of information/data exchanged between replicas is small (relative to the operating data-set size).

(*a*) *Mathematical Model*

In this subsection we develop a mathematical model that captures the primary components that make up the total runtime of a RE experiment. In an ideal scenario, the total time-to-completion an experiment would be equal to the concurrent runtime of the ensemble of replicas and there would be no overhead associated with the coordination of the replicas. If the ensemble contains N_R replicas, the total number of pairwise exchanges is N_X and the runtime of a replica to complete a defined number of time steps is T_{MD}, the total time-to-completion of the RE experiment T would be:

$$T = \frac{1}{p} \times (T_{MD} \times \frac{N_X}{\frac{N_R}{2}}) \tag{2.1}$$

where p is defined as the probability of a successful exchange (the probability of a successful exchange is not 1, the decision to accept an exchange or not is made using the Metropolis scheme (Metropolis et al. 1953)). However, any RE experiment will entail an overhead of coordination, job-submission & termination etc. Thus, the time T to complete the RE experiment where N$_X$ is the total number of exchanges is:

$$T = \frac{1}{p} \times [(T_{MD} \times \frac{N_X}{\eta}) + (T_X + T_W) \times \frac{N_X}{\eta}] \tag{2.2}$$

where, T_X is the time to perform a *pairwise* exchange and includes the following components, (i) time to find a partner (T_f), (ii) time to exchange/write/transfer files (T_{ex}) as well as (iii) managing state (e.g., in the advert-server) and book-keeping associated with replica pairing/exchanging (T_{mgmt}) (thus $T_X = T_f + T_{ex} + T_{mgmt}$); T_W is comprised of the synchronisation time spent by a replica waiting for other replicas to complete running (T_s), the time spent waiting to be restarted after each exchange and other associated costs (T_r). For example, in asynchronous-centralised case, T_r arises due to serialisation at the exchanging agent. Similarly T_{mgmt} arises due do different reasons – which may be related to implementation of the different RE algorithms. Finally, η is the number of independent exchange events occurring concurrently; for N_R replicas, this is typically $\frac{N_R}{2}$.

(*b*) *Synchronous Replica-Exchange*

Traditionally, RE algorithms have been implemented such that the exchanges have been synchronous. If the number of replicas is N_R, this leads to a *fixed* number of $\frac{N_R}{2}$ pairs of replicas are created. When *all* the replicas in the ensemble reach a pre-determined state (e.g. the Molecular Dynamics (MD) simulation completes a pre-determined number of steps), an exchange of temperatures between the paired replicas is attempted using the Metropolis scheme. If the exchange attempt is successful, parameters such as the temperature are swapped.

For synchronous RE formulation, all replicas must reach a pre-determined state (`done`), before exchanges are performed. T_W includes the time waiting for all the replicas in the ensemble to reach this state, the time spent waiting before the replicas are restarted after each exchange and any other miscellaneous costs.

A major limitation of this model is that replicas are paired in fixed groups and thus exchanges take place between pre-determined pairs of replicas. As a consequence, of pairs being determined before an exchange, although T_f is 0, this limits the number of possible exchange partners that are available for a given replica; this inhibits exchanges between replicas with non-nearest temperatures, and ultimately negates the possibility of crosswalks – where a crosswalk is said to occur when a replica originally with a low temperature reaches the upper temperature range and then returns to the lower temperature range.

In addition to limitations in modeling the physics, there are rigid replica-pairing is efficient only in homogeneous environments; for heterogeneous environments & systems, where resource availability and performance fluctuates, the need for synchronisation leads to slow-down & inefficiencies. We show how these limitations are overcome in the asynchronous (exchange) formulations of RE.

(*c*) *Asynchronous Replica-Exchange*

In asynchronous formulations of the RE algorithms (Li and Parashar 2007, Gallicchio et al. 2008), a replica does not have to wait for *all* other replicas to reach a pre-determined state. An exchange occurs whenever a replica reaches a pre-determined state. It then performs an exchange with another suitable replica in the ensemble. The reduction in *synchronisation* (wait) times comes at the cost of increased *coordination* costs. The specific values of the terms T_X and T_W, in Equation 2.2 differ from the synchronous formulations. Since each replica on completing a run has to find a *new* partner, $T_f \neq 0$. Additionally, T_W only includes the time spent waiting for the next replica to become available, any time spent waiting before the replicas are restarted after an exchange and any other miscellaneous costs.

3. Replica-Exchange Framework

An important motivation for this work is to design and implement a framework that provides the capability to implement and compare the performance of the different RE algorithms formulations at large-scales. In addition, it is important that the framework be

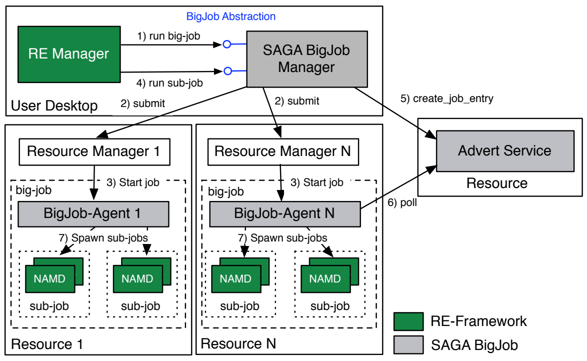


Figure 1: SAGA/BigJob Architecture

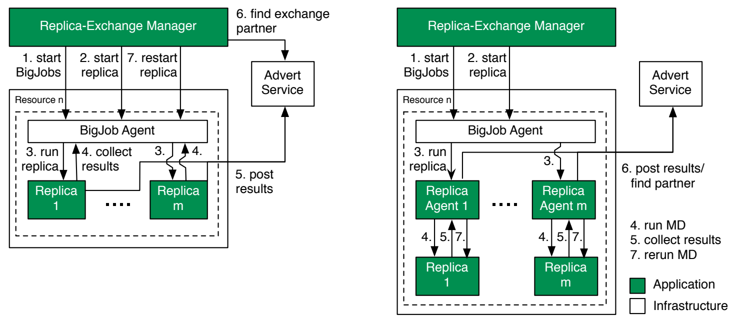
independent of the underlying infrastructure. It is useful to highlight that we differ from other RE implementations (e.g. Li and Parashar (2007)) in that we use *production grade* national and regional cyberinfrastructure, such as the US TeraGrid and LONI (Louisiana Optical Network Initiative 2011), using general purpose tooling and standard capabilities available on these production infrastructure. Additionally, our framework *natively* supports individual replicas that are MPI jobs. In this section we outline the architecture, implementation and the basic performance of the RE framework when used to implement the different formulations.

(*a*) *SAGA BigJob - A Pilot-job Framework*

We have demonstrated the usage of the SAGA-based Pilot-Job framework (Luckow et al. 2010) – called the BigJob, to run RE simulations across multiple, heterogeneous, distributed grid and cloud infrastructure (Luckow et al. 2009). SAGA (SAGA 2011) is an API to the basic functionality required to build distributed applications, tools and frameworks so as to be independent of the details of the underlying infrastructure. The various tasks that are carried out using the SAGA APIs include file staging, job spawning and the conduction of the exchange attempts.

Here we use the BigJob framework to efficiently request and manage computational resources for multiple replicas. Specifically, it enables the dynamical utilisation a range of infrastructures, i.e., it supports a scheme that does not depend on a static set of resources that are pre-defined at the time of workload submission.

Figure 1 shows the architecture of SAGA BigJob. It consists of three components: (i) the BigJob-Manager, (ii) the BigJob-Agent and (iii) the advert service which is a central key/value store which is used for communication between the BigJob-Manager and the BigJob-Agent. The BigJob-Manager submits the big-jobs to the resource manager and sub-job descriptions to the advert server. There is a separate BigJob-Agent for each big-job, even if there are multiple big-job on a given machine. Once a big-job becomes active, the BigJob-Agent retrieves the job descriptions from the advert server, allocates the required number of nodes and launches the sub-jobs on each resource. The BigJob-Agent continuously monitors the running sub-jobs and updates the sub-job states in the advert



(a) Central (b) Decentral

Figure 2: **Centralised vs. Decentralised Coordination:** Both central coordination style is used both by the synchronous RE and asynchronous (centralised) RE. In the asynchronous (decentralised) RE – the master is only required for initially setting up all replicas. The later coordination is done peer-to-peer via the Advert Service.

server. Once a sub-job finishes running, the nodes are freed and marked as available. The BigJob-Agent periodically polls the advert server for new jobs.

(*b*) *Replica-Exchange Manager*

The RE-Manager is the master process controlling the different components of the framework and aspects of the exchanges. The RE-Manager uses the SAGA BigJob-Manager; the actual tasks that the RE-Manager performs depend on which RE algorithm is being investigated. A replica-agent is a wrapper script that manages an individual replica – starts, facilitates the exchange of that replica, and restarts it.

The asynchronous RE algorithm can be implemented using a centralised or decentralised coordination scheme. In the former, the RE-Manager manages all the replicas and performs the exchanges; in the decentralised coordination implementation, multiple replica-agents take on that responsibility. It is worth noting that the synchronous RE algorithm has centralised coordination scheme. Interestingly, for decentralised coordination implementation, the RE-Manager is functionally similar to the SAGA BigJob-Manager.

We first explain how the synchronous and asynchronous (centralised) RE Manager works in the following section followed by an explanation of how the asynchronous (decentralised) RE-Manager/replica-agent combination works.

(i) *Synchronous and Asynchronous (Centralised) RE-Manager*

The control flow of a centralised RE scheme is shown in Figure 2(a), which can be used to understand both the asynchronous (centralised) RE and synchronous RE implementations. A replica can be in one of these three states: (i) `new` (submitted but not started), (ii) `running` and (iii) `done`. Once the BigJob(s) is active and replicas are `running`, the RE-Manager constantly queries the SAGA BigJob-Manager for the latest replica states. When the RE-Manager finds a replica that has finished running, it collects the energy and temperature of that replica by reading the output file. Once *all* the replicas have finished running, the RE-Manager performs the exchanges by swapping temperatures and writing new configuration files. The new configuration files are staged to the appropriate location. The RE-Manager then submits the replicas for restarting, and the SAGA BigJob-Manager restarts them. The RE-Manager keeps count of the successful exchanges, until the required number of exchanges are done.

The implementation of the asynchronous (centralised) RE-Manager, is different from the synchronous RE-Manager, in that, instead of waiting for *all* replicas to finish running before performing *all* exchanges, whenever the asynchronous (centralised) RE-Manager finds a replica that has finished running, it tries to find a partner to make an exchange. In order to find a partner, the RE-Manager goes over the list of all the replicas in the ensemble. If it finds a replica available it attempts the exchange. If a replica is not found available, the RE-Manager queries the SAGA BigJob-Manager for the latest replica states and updates its local list. It then loops over the list to find a replica that has finished running and a partner to exchange with that replica. If successful, the replicas are submitted to be restarted.

(ii) *Asynchronous (Decentralised) RE-Manager and the Replica-Agent*

Figure 2(b) shows the asynchronous (decentralised) RE control-flow. In the asynchronous (decentralised) implementation, in order to conduct the exchanges the RE-Manager launches multiple replica-agents (in lieu of replicas directly). Replica-agents then take control of replica start/re-start and exchange attempts. The replica-agents upon launch, run the replicas; a list of nodes that is used to carry out the MD run is passed to the replica-agent as an argument at startup. The replica-agent constantly monitors the replica, and when the replica finishes, it updates the advert server with the current state of that replica. It also reads the temperature and energy from the output files, and posts the values to the advert server. The RE-Manager is primarily responsible for keeping track and count of the number of exchanges performed; when the desired number of exchanges are done, the RE-Manager ends the experiment.

A replica can be in one of the these four states: (i) `running`, (ii) `done`, (iii) `pending` (for an exchange) and (iv) `complete` (exchange has been completed). When a replica reaches a pre-determined state it (the replica-agent acts as the proxy for the replica) transitions from `running` to `done`. It initiates the search for a partner, and scans the list of replicas randomly, so as to avoid contention if multiple replicas have initiated a search for a partner. If a replica finds another replica available, it reverifies that their states as still in `done` state; if both are in `done` state, the exchange can proceed and their states are set as `pending`. When *reverifying* the state of the replicas, if the state of the initiating replica has already been changed (to `pending`), the current exchange attempt is aborted; if not, the exchange attempt proceeds.

After the exchange is performed (the temperature of both replicas in the advert server has been changed), their states are set as `complete`. Here the state `complete` is a marker which tells the replica-agent that the exchange has been made, the configuration files are in place and to restart the replica. The associated replica-agents write new configuration files with updated temperatures, and restart their replicas. The replica-agent that initiated the exchange increments the exchange count. The RE-Manager constantly queries the advert server for the latest exchange count and when all exchanges have been made, it stops the experiment.

4. RE Framework: Implementation and Performance

In §3 we presented the basic components of the RE framework and discussed the control flow for the three RE algorithm formulations. In this section we provide further details of

	synchronous	**asynchronous (decentralised)**	**asynchronous (centralised)**
T_{MD}	71 s	71 s	71 s
T_W	2.8 s	0.7 s	1.1 s
T_s	2.8 s	0 s	0 s
T_r	0 s	0.7 s	1.1 s
T_X	0.6 s	7.2 s	0.8 s
T_f	0 s	6.4 s	0.2 s
T_{ex}	0.4 s	0.6 s	0.4 s
T_{mgmt}	0.2 s	0.2 s	0.2 s
T	**1003 s**	**631 s**	**811 s**

Table 1: Average values of terms in Eqn 2.2 for the three RE algorithms. T_{MD}=time replica takes to complete 500 timesteps; T_W=time waiting for other replicas and to be restarted; T_s=synchronisation time; T_r=time waiting to be restarted; T_X=time to make a pairwise exchange; T_f=time to find/lock a partner; T_{ex}=time to write/transfer files; T_{mgmt}=updating states after exchange; T=total time-to-completion.

the working of the RE framework, as well as basic characterization of the performance of the RE framework for the three RE algorithm formulations.

For the basic characterisation, the following experimental configuration is used: (i) Infrastructure: Our experiments are performed on LONI and Teragrid shared resource *Queen-Bee (QB)*. A highly scalable, parallel MD code – NAMD (Phillips et al. 2005), is utilised to perform the MD simulations for each replica (although, it is important to mention that any other MD or Monte Carlo code could be used just as simply and effectively with the RE framework). (ii) Replica-Exchange Configuration: The total number of replicas (N_R) in the ensemble are 32 and the total number of pairwise exchanges (N_X) is 128. As the ensemble of replicas are run concurrently, 16 pairwise exchanges are possible after each concurrent run. Thus, each replica on average is restarted 7 times. Each replica is configured to run 500 time-steps and is allocated 16 processors. One BigJob of size 512 processors is requested. On average each 500 time-step run takes 71 s. For all implementations, in the event of a successful exchange, jobs are restarted (Luckow et al. 2009) with new temperature values. In the case of an unsuccessful exchange, jobs are restarted without exchanging the configuration. (iii) The physical system that we use as benchmark is the Hepatitis-C Virus that was examined in Ref. Luckow et al. (2009).

The different RE algorithms were repeated multiple ($\approx$ 10) times, during different load factors; therefore specific queue wait times can be approximated to be similar. Additionally, as we will be interested in understanding the scale-up and scale-out properties of synchronous and asynchronous RE, we do not consider queueing times. As explained in § 2 (a), the relative performance of RE implementations is primarily determined by the waiting time T_W and the time for conducting the exchange T_X. In the following we analyse the average values for T_W and T_X for each replica pair. Table 1 summarizes the results. We will discuss each case in the following sections.

(*a*) *Synchronous RE*

In the synchronous RE implementation replicas are started sequentially, i. e. there is a delay between the startup of the first and the last replica. Also, the post-processing of each replica run, i. e. updating the state, marking nodes free, the stage-out of the output file, is done sequentially. The longer of the two determines the overall time spent waiting

(T_s) for other replicas. In this case the post-processing time is the larger, and on average, it takes $1.4\,s$ to process a replica that has completed running. Thus, T_s, which is defined for a pair of replicas, is $2.8\,s$. For an ensemble of 32 replicas (with 16 pairs) the delay between the first and last replica transitioning to `done` state adds up to $44.8\,s$. The waiting time at the BigJob-Agent is subsumed by the synchronisation time, thus the BigJob-Agent is effectively always ready to start replicas, thus, T_r is 0 for synchronous RE.

T_X comprises of three sub-components: T_f, T_{ex} and T_{mgmt}. In this scenario $T_f = 0\,\mathrm{s}$ due to the fact that there is fixed pairing. The updating and stage-out of the configuration files, is observed to be approximately $0.2\,s$ per replica and thus, $T_{ex} = 0.4\,s$ per replica pair (the transfer of the input files is done sequentially). T_{mgmt} is the time required by the RE-Manager to post job description to the advert server. On average T_{mgmt} amounts to $0.1\,\mathrm{s}$ per replica, i. e. $0.2\,\mathrm{s}$ per replica pair. Thus T_X is: $0 + 0.4 + 0.2 = 0.6$ s.

Although there are $\frac{N_R}{2}$ concurrent pairs, the exchange at the RE-Manager is carried out sequentially; thus the effective number of concurrently exchanging pairs is 1. Substituting the above values in equation 2.2, we get:

$$T = \frac{1}{p} \times [(71 \times \frac{128}{16}) + (0.6 + 2.8) \times 128] = \frac{1}{p} \times 1003\,s. \tag{4.1}$$

(*b*) *Asynchronous (decentralised) RE*

A fundamental difference between synchronous and asynchronous formulations of RE is in synchronisation barrier for the replicas before exchanges, i.e., the former has a barrier, the latter does not. So although, $T_s = 0$ for asynchronous formulations, this comes at the cost of a higher T_f, the lack of a synchronisation barrier leads to a more involved implementation to dynamically pair replicas.

In the asynchronous (decentralised) RE, the replicas are managed individually by replica-agents. The replica-agent constantly monitors all replicas. If it finds a replica that completed its run, it updates its state in the advert server, which takes $0.1\,s$. The replica-agent retrieves the energy and temperature from the output files, each of which takes $0.2\,s$. It then posts both these values to the advert server. The waiting times to retrieve these values is thus $0.1 + 0.2 \times 2 + 0.1 \times 2 = 0.7\,s$, so $T_r = 0.7\,s$. Note that here T_r involves waiting at the replica-agent and not the BigJob-Agent (like the centralised case). Therefore the value of T_W $0.7\,s$.

T_X is primarily determined by T_f, i. e. the time necessary to lock an exchange partner. When searching for a replica randomly, on average it takes $\frac{N_R}{2}$ attempts to find a replica that is in the `done` state. However, finding a replica in the `done` state is not enough to ensure an exchange attempt. Given that there are several "active exchanges" being attempted, often the reverify step leads to an aborted exchange attempt; a reverify step must occur to ensure there has been no change in the states of either of the two replicas involved. The exact number is a random variable, determined by the number of replicas, the distribution of states and whether the attempt to find a replica is random or sequential. Empirical observation suggests that between 2-4 find and reverify attempts before an exchange is attempted. But in general, replicas contend with each other to lock a partner for exchange; this gets worse with increasing number of replicas, i.e., T_f increases with increasing N_R. Specifically, for the random access case, we find T_f is $2 \times N_R \times 0.1$ (where 0.1 is the typical time to set/get a value to/from the advert server). The process of exchanging the states and temperatures via the advert server and writing a new configuration file takes, $T_{ex} = 0.1 \times 4 + 0.2 = 0.6\,s$. T_{mgmt} is the time it takes to update the state in the advert server, which is $0.2\,s$. Thus, T_X is $6.4 + 0.6 + 0.2 = 7.2\,s$. It should be noted that T_X is highly dependent on the actual implementation. While the current implementation is kept simple on purpose, this value can be improved in a more sophisticated implementation.

As there are $\frac{N_R}{2}$ concurrent pairs, substituting the above values in equation 2.2, we get:

$$T = \frac{1}{p} \times (71 \times \frac{128}{16}) + \frac{(7.2 + 0.7) \times 128}{16} = \frac{1}{p} \times 631\ s. \tag{4.2}$$

(*c*) *Asynchronous (centralised) RE*

As in the decentralised case, the asynchronous centralised RE algorithm does not require synchronisation between *all* replicas in order to transition a replica-pair from `running` to `done` state and $T_s = 0$ by definition. Using a centralised implementation the time to find an exchange partner (T_f) can be reduced; however, this comes at a tradeoff that there is some contention at the master. Also, we observed some delays at the BigJob agents during the startup of the replicas sub-jobs mainly due to the fact that the BigJob-Agent is single-threaded and thus, is busy processing other replicas after their termination. Specifically, the time-to-submit a replica-pair to the BigJob-Manager, i. e. two replica sub-jobs, is in the asynchronous (decentralised) case in average 1.1 s. This is 0.5 s longer than in cases without contentions at the BigJob-Agent – in these case the submission of two sub-jobs requires only 0.6 s.

In contrast to the synchronous case, T_X for the asynchronous centralised case has a T_f component since replica pairs are dynamically determined and not fixed. T_f depends on the overall number of replicas (N_R), which determines the number of records the RE-Manager has to scan in order to find an available replica. On average the RE-Manager must search through $N_R/2$ replicas before it finds a partner. Although the search for a replica is random like the decentralised implementation, there is no need for reverifying, as due to centralised control there is no contention in replica pairing, i.e., exchanges are made by the RE-Manager and only one exchange takes place at a time. An advert query for a replica state takes 0.01 s. Note this is a factor of 10 less than the decentralised implementation where there was a connection to the advert server for every replica-agent; here there is a single connection. For 32 replicas, the RE-Manager requests on average 16 other replica states before it finds a partner; thus, T_f is in total $0.01 \times 16 = 0.2\,s$. Both T_{ex} and T_{mgmt} are same as in the synchronous case, i. e. T_X is thus: $0.2 + 0.4 + 0.2 = 0.8\,s$.

As in the synchronous case the effective number of concurrently exchanging pairs is 1 due to the fact that exchanges are sequentially carried out by the RE-Manager. Substituting the above values in equation 2.2, we get:

$$T = \frac{1}{p} \times [(71 \times \frac{128}{16}) + (0.8 + 1.1) \times 128] = \frac{1}{p} \times 811\ s. \tag{4.3}$$

5. Scale-Up and Scale-Out: Experiments and Results

To evaluate the scaling properties of the different RE algorithms and implementations, we conducted several experiments on TG and LONI resources. We initially increased the number of replicas while keeping the number of machines constant ("scale-up"); then for the asynchronous-centralised case, for a given number of replicas, we varied the number of distributed machines utilised ("scale-out"). In this section we outline the experiments conducted. Results of these experiments establish the advantages of asynchronous formulations for both scaling-up and scaling-out.

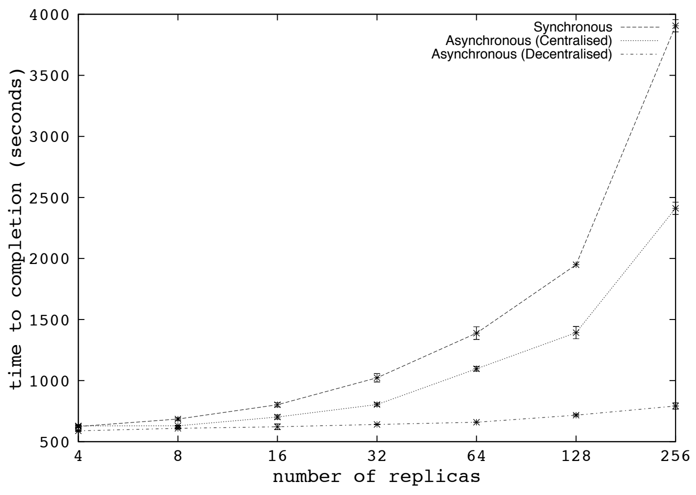


Figure 3: **Scale-Up Performance for 8 to 256 Replicas:** The graph shows the runtimes for the different RE implementations. The ratio between the number of exchanges and number of replicas is kept constant. Each replica is assigned 16 processors and run 500 timesteps. The asynchronous decentralised RE implementation shows the best scaling behaviour. Both centralised RE versions scale less well mainly due to the limitations of the single master, which becomes a bottleneck.

(*a*) *Scale-Up*

Experiments: To understand the scaling behaviour of the three cases, we compute the T for 4, 8, 16, 32, 64, 128 and 256 replicas, for 16, 32, 64, 128, 256, 512 and 1024 exchanges respectively. This fixes the ratio of the number of exchanges to the number of replicas ($\frac{N_X}{N_R}$) to 4. Each replica is configured to run 500 time-steps before an exchange is attempted, and is allocated 16 processors. Experiments up to 128 replicas are performed on *QueenBee*, while experiments with 256 replicas are done on *Ranger*; this is because *QueenBee* only allocates a maximum of 2048 processors per job request. We have normalised the the data to factor in the difference in performance of *Ranger* and *QueenBee*.

Results: Figure 3 shows the results obtained by running the scale-up experiments mentioned. As a consequence of the ratio of the number of attempted exchanges to of replicas being a constant, the T_{MD} term across the different values of N_R is essentially the same; hence comparison between different cases will reveal differences in the coordination cost (T_W & T_X). Thus, as N_R increases, the variation in T is due to the coordination component – terms T_X and T_W in Equation 2.2. The increase, however, is not uniform across the three implementations: it is largest for synchronous RE, and the least for asynchronous (decentralised) RE. We analysed T and the values of its components for 32 replicas in § 4.1; we use that analysis as the basis to understand the scale-up behaviour of the three cases.

In the synchronous RE algorithm, there is an explicit synchronisation of all replicas; thus early replicas wait for other replicas to finish. As can be seen from Table 1, T_s is a major component of the T. As N_R increases, the number of exchanges at a given exchange step increases; consequently the total coordination time at each exchange step increases. For a value of N_R of 64 and N_X of 256, using Equation 2.2, the coordination time is

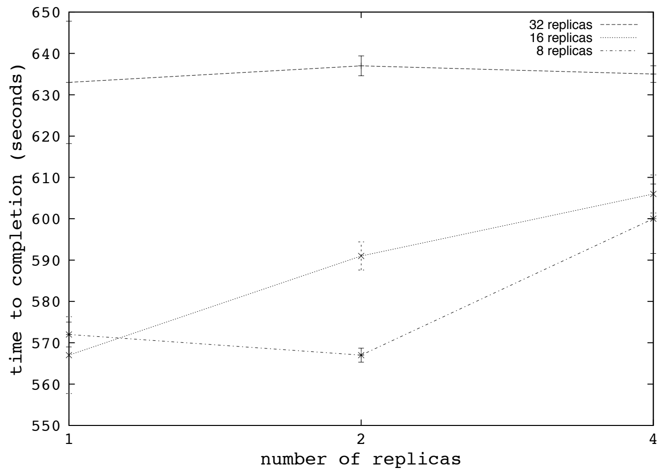


Figure 4: **Scale-out performance for 8, 16 and 32 replicas, asynchronous (centralised):** As the number of machines increases, the time-to-completion increases in general mainly due to higher exchange costs (T_X) caused by e. g. remote file copies and remote advert service updates.

$(0.6 + 2.8) \times 256 = 870.4$. The difference in coordination times (435.2 s) accounts for the difference in T for 32 and 64 replicas in the Figure 3. A similar analysis was performed for different replica and exchange counts and the values obtained are in agreement with the empirical data in Figure 3.

For the decentralised implementation, T_f has a strong N_R dependence. We find T_f is $2 \times N_R \times 0.1$ (where 0.1 is the typical time to set/get a value to/from the advert server). For example, in Equation 4.2 we see that for 64 replicas, T_f is approximately $2 \times 64 \times 0.1 = 12.8$ s. Thus the difference in T_X when N_R is 64 versus 32 is 51.2s, which accounts for the bulk of the increase in overall T when N_R changes from 32 to 64. The variation in T for other values of N_R can be accounted for by changing values of T_f.

In contrast to the decentralised implementation, T_f in the asynchronous centralised case is only weakly dependent on N_R; however, since the RE-Manager makes the exchanges serially, we still see an increase in the time-to-completion with increasing N_R. For $N_R = 64$ and N_X 256, using Equation 4.3 leads to a new coordination $(T_W + T_X)$ time of $(0.8 + 1.1) \times 256 = 486.4\,s$, up from 243.2 for $N_R = 32$. This change accounts for a large component of the difference in T as N_R goes from 32 replicas to 64 replicas, as shown in Figure 3. The actual difference observed in T from 32 to 64 replicas is 293 s. We have verified this for different number of replicas and found it to be consistent.

In addition to the above explicable changes arising from different coordination costs, as N_R increases different nodes of a machine tend to perform differently, thus influencing the time-to-completion of different replicas, but not always monotonically. As N_R increases, the the time to start and synchronise replicas typically increases.

(*b*) *Scale out - Asynchronous (centralised)*

Experiments: To evaluate the performance of asynchronous (centralised) implementation when scaling out across many machines, we used the following TeraGrid and/or LONI resources: *QueenBee, Eric, Louie* and *Oliver*. The experiments conducted are 8, 16 and 32 replicas making 32, 64 and 128 exchanges. These exchanges are repeated on 1, 2 and 4 machines while distributing the number of replicas equally on each machine. It is important to note that all experiments are conducted using four BigJobs, irrespective of the number of machines used; varying the number of BigJobs as well as the ratio of replicas per BigJob effects overall performance. Another important point to note is that only the experimental runs where all the four BigJobs become active within $30\,s$ of each other on submission to the resource manager are considered and included in the results. This is to remove the queue wait time from the equation and focus on the runtime. Each experiment is repeated 5 times.

Results: The results obtained are shown in Figure 4. The time-to-completion T increases moderately or remains constant with higher number of machines, which indicates that the scale-out behaviour is quite good. As discussed in section 4, the asynchronous (centralised) RE algorithm does not involve any synchronisation cost (T_s). Thus the loose coordination between replicas at the exchange stage helps provides the flexibility to use multiple resources, which would not be possible if there was tighter coordination between replicas at the exchange stage. The main contributor of the small increase is T_X are the higher costs for remote file transfers and remote advert updates; fluctuations in remote operations also account for the somewhat higher variations in the T.

6. Conclusion

Following theoretical underpinnings provided in Ref. (Li and Parashar 2007, Gallicchio et al. 2008), in this paper we investigate *traditional* and *advanced* replica-exchange algorithms at unprecedented scales. An important motivation for this work has been to implement a framework for the RE class of algorithms that can use general purpose and standard capabilities available on production infrastructure, such as, the Teragrid and LONI. Additionally, our framework uses a flexible pilot-job implementation, which enables effective resource allocation for multiple replicas.

Results shown in figures 3 and 4 indicate that using algorithmic formulations that impose less tight coordination constraints enable both good scale-up and scale-out behaviour. Algorithms based on asynchronous coordination are typically more difficult to implement than synchronous ones; however, we find that even with a simple, non-optimised prototype of the RE-framework, the advantages of asynchronous formulations soon out weight the synchronous formulations, i.e., as N_R increases the performance of asynchronous RE beats that of the synchronous RE.

Our analysis shows that a fundamental trade-off is between the lower cost of replica synchronisation at the exchange stage that asynchronous formulations provide, versus the higher cost of permitting dynamic replica pairing. In an attempt to investigate an optimal trade-off between these factors, and to demonstrate the advantages of asynchronous RE, we implemented a centralised version of the asynchronous RE with a lower cost of dynamical pairing than in the decentralised implementation. Our initial results show promising scale-out behaviour, but more work is required to separate and understand fundamental algorithmic advantages from implementation specific issues.

Acknowledgements

This work is part of the Cybertools (http://cybertools .loni.org) project and primarily funded by NSF/LEQSF (2007-10)-CyberRII-01. Important funding for SAGA has been provided by the UK EPSRC grant number GR/D0766171/1 (via OMII-UK) and HPCOPS NSF-OCI 0710874. This work has also been made possible thanks to computer resources provided by TeraGrid TRAC TG-MCB090174 and LONI resources.

References

Gallicchio, E., Levy, R. M. and Parashar, M. 2008, Asynchronous replica exchange for molecular simulations, *Journal of Computational Chemistry* **29**(5), 788–794.

Hansmann, U. 1997, Parallel Tempering Algorithm for Conformational Studies of Biological Molecules, *Chemical Physics Letters*, Vol. 281, pp. 140–150.

Li, Z. and Parashar, M. 2005, Comet: A scalable coordination space for decentralized distributed environments, *Proceedings of the Second International Workshop on Hot Topics in Peer-to-Peer Systems*, IEEE Computer Society, Washington, DC, USA, pp. 104–112.
URL: *http://portal.acm.org/citation.cfm?id=1090948.1091381*

Li, Z. and Parashar, M. 2007, Grid-based asynchronous replica exchange, *GRID '07: Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, IEEE Computer Society, Washington, DC, USA, pp. 201–208.

Louisiana Optical Network Initiative 2011. http://www.loni.org/.

Luckow, A., Jha, S., Kim, J., Merzky, A. and Schnor, B. 2009, Adaptive Replica-Exchange Simulations, *Royal Society Philosophical Transactions A* .

Luckow, A., Lacinski, L. and Jha, S. 2010, Saga bigjob: An extensible and interoperable pilot-job abstraction for distributed applications and systems, *The 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A. and Teller, E. 1953, Equation of State Calculations by Fast Computing Machines, *The Journal of Chemical Physics* **21**(6), 1087–1092.

Phillips, J., Braun, R., Wang, W., Gumbart, J., Tajkhorshid, E., Villa, E., Chipot, C., Skeel, R., Kale, L. and Schulten, K. 2005, Scalable molecular dynamics with NAMD, *Journal of Computational Chemistry* **26**, 1781–1802.

SAGA 2011.
URL: *http://saga.cct.lsu.edu*

Sugita, Y. and Okamoto, Y. 1999, Replica-Exchange Molecular Dynamics Method for Protein Folding, *Chemical Physics Letters*, Vol. 314, pp. 141–151.