Reducing Complexity in Management of eScience Computations

Per-Olov Östberg*, Andreas Hellander‡, Brian Drawert‡, Erik Elmroth*, Sverker Holmgren† and Linda Petzold‡
*Dept. of Computing Science, Umeå University, SE-901 87 Umeå, Sweden. {p-o,elmroth}@cs.umu.se
‡University of California, Santa Barbara, CA 93106-5070 Santa Barbara, USA. {andreash,bdrawert,petzold}@cs.ucsb.edu
†Uppsala University, SE-751 05 Uppsala, Sweden. {sverker}@it.uu.se

Abstract—**In this paper we address reduction of complexity in management of scientific computations in distributed computing environments. We explore an approach based on separation of computation design (application development) and distributed execution of computations, and investigate best practices for construction of virtual infrastructures for computational science - software systems that abstract and virtualize the processes of managing scientific computations on heterogeneous distributed resource systems. As a result we present StratUm, a toolkit for management of eScience computations. To illustrate use of the toolkit, we present it in the context of a case study where we extend the capabilities of an existing kinetic Monte Carlo software framework to utilize distributed computational resources. The case study illustrates a viable design pattern for construction of virtual infrastructures for distributed scientific computing. The resulting infrastructure is evaluated using a computational experiment from molecular systems biology.**

I. Introduction

For scalability, most applied science computational software would benefit from utilization of parallel computing resources, expanding their capabilities as eScience (computationally intensive science carried out in highly distributed network environments) applications. However, many applications in the scientific community are first developed for local computation. Scaling scientific computations from local environments to dedicated computing environments is complicated, as it alters the way computations are performed and often requires changes to computational methods to efficiently utilize parallel resources. Scaling computations to distributed computing environments, e.g., grid or cloud environments, further complicates the process as it again alters the way resources are utilized and now also introduces additional factors such as distributed data and failure management, resource set heterogeneity, and cross-domain security issues.

Computational researchers and developers of scientific software are often cognizant of parallelization issues, but lack detailed knowledge of the issues involved in computational enactment in distributed environments. The task of adapting existing desktop software to execute in a grid or cloud environment without the aid of a field specialist can hence be an insurmountable task, especially in small software projects composed of application specialists. Thus, design patterns and integration tools that help to overcome such barriers can be expected to have a great impact in many areas of eScience by expediting migration of existing computational applications to scalable computational resource sets.

In this work we discuss scaling of scientific computations to dedicated and distributed computing environments using an approach based on separation of computation design and execution. The vision of this work details virtual infrastructures for computational science that abstract and virtualize the processes of executing, managing, and monitoring scientific computations on distributed resource sets. The model used is focused on preservation of the roles of the tools involved in scientific computing; specialized applications should still allow end-users to interface and utilize them the way they do for local computations, and generalized computing infrastructure components should preserve their operation while being extended to support distribution of computations. This methodology is realized in two steps; development of virtual infrastructure tools that abstract and automate tasks related to execution of computations, and intelligent integration of these tools with existing computing tools and applications.

To illustrate the methodology, we build on preliminary work presented in [13] and present a case study where we extend the capabilities of an existing open-source computational software framework from the systems biology community to utilize cluster and grid resources. As a result of this work we present StratUm, an extensible toolkit for migration of computational applications to distributed resource environments.

In the case study, computational resources are accessed through the Grid Job Management Framework (GJMF) [12], a grid job management tool that manages infrastructure tasks such as brokering, monitoring, and control of job executions on grid resources. The existing environment is extended with capabilities for data management, more efficient communication, security, and notification models, as well as a set of native client APIs. The extended environment, StratUm, is then integrated with URDME [3], a Matlab-based environment designed for computational experiments, using a set of custom computational clients that allow the core parts of the computational application to remain unchanged. To highlight some of the trade-offs in these kinds of integrations, the efficiency of the resulting environment is then evaluated in a computational experiment from the systems biology literature.

Efforts similar to StratUm exist and include, e.g., Falkon [15], a lightweight task execution framework designed for Many Task Computing [14]. While Falkon and StratUm are similar in use of custom protocols and service interfaces, Falkon is designed for high submission throughput whereas

StratUm is focused on providing high abstraction levels for computation. The GridSAM [10] grid job submission system is similar in design to StratUm in that it is standards-based and abstracts underlying resource managers through service-based interfaces. GridSAM also builds on a staged event-driven architecture (SEDA) [17] that is similar to the job processing pipeline of StratUm and the GJMF [11].

The StratUm native client APIs are conceptually similar to the Simple API for Grid Applications (SAGA) [9], which is an API standardization initiative that, like StratUm and the GJMF, aims to provide a unified interface to distributed resource integration. The design philosophy of the SAGA API differs from StratUm in that StratUm focuses on high-level automation and facilitation of integration customization. Performance evaluations comparing StratUm to SAGA implementations are a subject for future work. For reasons of brevity, the list is naturally far from complete. In general, the approach of this work differs from most in that StratUm aims to abstract resources from multiple types of infrastructures and facilitate development of customized integration tools that support application-specific use cases. For more exhaustive treatment of grid job management mechanisms we refer to [12].

On the application side, enactment of stochastic simulations in a cloud environment have been demonstrated using the domain-specific language Neptune [1]. Although both StratUm and Neptune strive to be mechanisms by which scientific computations are enacted on remote distributed computing resources, the StratUm approach is based on abstracting job management complexity and facilitating development of application-specific integration tools, while Neptune is focused on resource management while enacting computation on cloud computing platforms. For a related stochastic modeling framework, a dedicated virtual infrastructure has been proposed for simulations with MCell in grid environments [2].

The rest of this paper is organized as follows. Section II presents our approach and details the design and components of the StratUm toolkit. Section III illustrates use of the toolkit in the context of an integration case study. This section contains an introduction to the computational problem, and a brief overview of the application used in the case study. To characterize and illustrate the capabilities of the toolkit, Section IV presents an evaluation of the integrated system and Section V contains a discussion of the methodology used and the experiences gained from developing and using the toolkit. Finally, Section VI concludes the paper.

II. Stratified Resource Abstraction

The computational requirements of eScience applications often require computational capacity well beyond the compute and storage capabilities of individual computational resources. To increase available computational capacity, eScience applications are often migrated to dedicated resource environments, e.g., HPC clusters or distributed grid environments. Dedicated computing environments introduce new usage patterns that include computation management tasks such as distributed data management and remote computation monitoring, which can amount to substantial overhead in resource utilization. In migration, the complexity of such management tasks are often underestimated, resulting in problem- or environment-specific solutions for distribution of computations.

In our approach we argue that suitable and reusable design patterns that facilitate integration and migration of eScience applications to distributed computing environments will require high-level abstractions on both the job management level as well as on the level of the computational application. In line with this reasoning, we utilize an approach based on the clean separation of (application-side) computation design and (infrastructure-side) computation management. In our approach we design tools that operate in the borders between these two areas and provide generic computation management functionality coupled with application-specific integration environment tools, e.g., smart infrastructure clients.

In this section we present StratUm, a toolkit for migration of computational applications to distributed computing environments. To relate the toolkit to earlier contributions and its operation context, we first briefly introduce the Grid Job Management Framework (GJMF), an existing framework for management of scientific computations in grid environments.

A. The Grid Job Management Framework (GJMF)

The Grid Job Management Framework (GJMF) [12] is a grid environment computation management toolkit designed as a loosely coupled service-oriented architecture. Constructed as a hierarchically ordered set of services, the GJMF has an abstractive design where higher level services aggregate the functionality of lower level services and offer increasingly advanced functionality and automation. The services of the GJMF offer concurrent access to multiple grid middlewares through a unified set of job management interfaces.

In addition to a range of job submission, brokering, control, and monitoring interfaces, the framework offers advanced service features such as dynamic fail-over capabilities, dynamic recomposition of the framework, and a set of customization points that allow third parties to modify the internal workings of the framework without altering the framework design. The GJMF is implemented in Java, is constructed using the Globus Toolkit 4 (GT4) [7], and utilizes Web Service Resource Framework (WSRF) notifications for state coordination.

B. The Stratified Resource Abstraction Toolkit (StratUm)

While the GJMF provides a generic and flexible framework for computation enactment in grid environments, it inherits some technical features that limit the integration flexibility of the framework. For example, the GJMF exposes functionality through web service interfaces and utilizes WSRF notifications for distributed state updates, which requires computational applications to host service containers to fully benefit from the notification model. Additionally, the GJMF builds on the security model of the Globus Toolkit v4, and utilizes x509 certificates and key pair proxies for authentication of end-users. While this is a robust security model, utilization of these

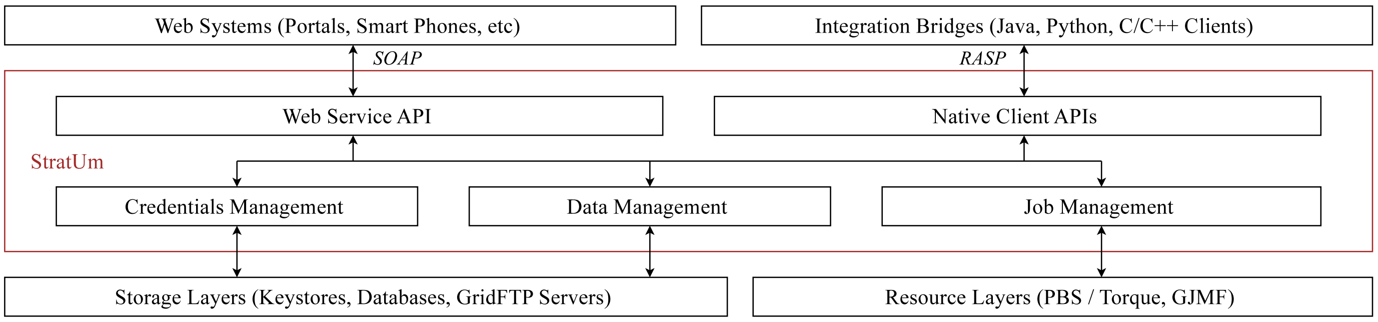
Web Systems (Portals, Smart Phones, etc)
Integration Bridges (Java, Python, C/C++ Clients)
SOAP
RASP
Web Service API
Native Client APIs
StratUm
Credentials Management
Data Management
Job Management
Storage Layers (Keystores, Databases, GridFTP Servers)
Resource Layers (PBS / Torque, GJMF)

Fig. 1: StratUm architecture. Toolkit functionality exposed as services through a custom protocol and (optional) web services. Web service and native client APIs abstract service communication, credentials, data, and job management complexity.

kinds of mechanisms can be perceived as complicated by end-users with limited experience of advanced security models.

The Stratified Resource Abstraction Toolkit (StratUm) is designed to constitute an abstractive layer for computation management. Its name comes from the geologic term *stratum*, a layer of sedimentary rock, while the *Um* emphasis of the name refers to the toolkit place of origin, Umeå University. As illustrated in Figure 1, the functionality of the toolkit can be organized into three main areas: credentials, data, and job management. Access to the functionality is provided through two types of APIs, web service interfaces and native client APIs. Communication with the components of the framework is routed through two types of protocols: generic SOAP and a custom hybrid protocol called the Resource Access and Serialization Protocol (RASP). The abstractions provided by the StratUm toolkit can be categorized as follows:

1) Credentials Management: The StratUm toolkit extends the GSI-based security model of the Globus Toolkit and offers mechanisms for storage, management, translation, and delegation of authentication credentials. The security model is based on a Public Key Infrastructure (PKI) where key pairs are associated to end-user identities via certificates. StratUm stores credentials securely on the service side using keystores and databases. Separate service APIs are offered for management of security credentials and identities. Using a mechanism similar to the GT4 delegation service, StratUm creates proxies (temporary delegated credentials) for job submissions. In addition to sending proxies directly to StratUm, the StratUm client APIs also support authentication via username-password tokens, which are sent using challenge-response authentication schemes built into the RASP protocol.

Use of advanced security models can impose steep learning curves and complicate access to computational infrastructures. The StratUm services and client APIs are designed to reduce the complexity of security-related tasks, e.g., authentication (client APIs provide mechanisms for authentication using username-password and challenge-response schemes as well as key pair-based authentication), authorization (temporary key pairs and certificates are automatically generated and used for resource-side job submissions without end-user interaction), and credentials management (tools are provided for client-side key pair and certificate generation and management as well as for secure transmission and server-side storage of authentication credentials). The design of the StratUm security mechanisms allows end-users to access infrastructure capabilities with a minimum of security complexity and overhead.

2) Data Management: The GJMF does not actively participate in any type of data transfer or storage. The GJMF coordinates data staging through management of standardized Job Submission Description Language (JSDL) documents, and relies on underlying Grid middleware to manage data transfers. Data storage is managed by external storage systems, and no validation of data availability is performed by the framework.

Management and transmission of data sets of scientific applications is complicated in distributed computing environments. To reduce the complexity of managing data in computation, Stratum provides a set of mechanisms for simple transmission and intermediate storage of data. On the service side, StratUm can be configured to host dedicated storage solutions, e.g., GridFTP servers, which are abstracted and monitored in the same way as external storage solutions by the toolkit. Data transfer interfaces are part of the client APIs and efficient transfer of data files to and from the toolkit storage systems is offered through RASP. While RASP supports (chunked and enveloped) transmission of large binary data payloads, it should be noted that this optional feature is to be regarded as a convenience mechanism, designed to reduce the threshold for migration of applications. For advanced users, and maximization of transmission performance and computational throughput, use of dedicated third party transmission utilities such as GridFTP is recommended. The StratUm data management mechanisms are entirely optional and simply convenience mechanisms designed to reduce the complexity of transmission and temporary storage of data.

3) Job Management: The job management capabilities of StratUm include job submission, monitoring, and control interfaces, as well as basic support for execution of static workflows. Job descriptions use the standardized JSDL language, and all job instantiations are monitored and managed by services in the toolkit. Job status updates are propagated through a notification mechanism built into the RASP protocol, and clients can register for updates using a publish-subscribe scheme. The StratUm workflow model employs a recursive workflow definition that details workflows to be ordered sequences (including parallel constructs) of tasks and sets of dependencies between tasks. Workflow nodes (tasks) are defined as abstract computational tasks and can be realized as individual jobs or other workflows. This definition allows workflows to consist of sets of tasks and sub-workflows, which allows end-users to use a single interface to describe and monitor all types of tasks (be it single jobs or large workflows).

The StratUm workflow model assumes that sequences of tasks that can be deterministically described in advance are represented in static workflow descriptions, while dynamic workflows (e.g., where branching of the task sequences are determined by results of prior tasks) are described in programming languages and coordinated by agents external to the toolkit. As demonstrated in the case study, this model provides the functionality required for coordination of execution of jobs in distributed resource environments. Conceptually, StratUm workflows may be viewed as directed acyclic graphs, where graph nodes contain computational tasks, i.e. jobs or workflows. Dependencies between nodes can be defined explicitly (control flow) or derived from implicit task constraints (e.g., data flows derived from data dependencies). The StratUm job dispatchment mechanisms are designed to be data aware, and will not process jobs before all required data are available.

4) Client APIs: The StratUm toolkit exposes functionality through two types of client APIs: web service interfaces and native client APIs. The APIs expose a basic functionality set that comprises operations for authentication, credentials management, (optional) data storage and retrieval, job submission, control and monitoring, as well as log management. To reduce the complexity of integration with the toolkit, the APIs are designed to be as simple as possible. For example, the task API offers operations to add, remove, and check/listen to the status (including logs) of tasks. All APIs abstract the complexity of communicating with distributed resources.

The web service interfaces communicate over SOAP, while the native client APIs communicate over RASP. RASP is designed for integration with the StratUm APIs, and is a hybrid wire-transport protocol that offers efficient serialization and parsing of message data. Messages are represented in tree format, where tree nodes contain hash maps that map text-resolved tags to binary payloads. RASP defines a message-oriented protocol stack that supports advanced protocol features such as in-message binary data payloads, secure authentication schemes, and transparent notification delivery. These advanced features are exposed through the native client APIs and fully abstracted in the toolkit programming model.

The purpose of providing native, multi-language client APIs is to reduce the learning curve of using the toolkit and to facilitate migration of scientific applications to distributed resource environments. The StratUm client APIs are designed to be as simple to use as possible and aim to abstract communication complexity whenever possible. The StratUm native client APIs are available in Python, Java, and C/C++, and are complemented with a set of clients that demonstrate the use of the APIs and constitute a set of tools that can be used to run jobs on distributed resources through StratUm.

III. Case Study

In this section we present a case study of a computational infrastructure for stochastic simulation of biochemical reaction networks. The resulting system is based on two existing tools: URDME, a public domain software package for stochastic reaction-diffusion simulation, and GJMF, a middleware-agnostic grid job management framework. Between these two we place a customized StratUm-based integration architecture designed to abstract and reduce integration complexity for both applications and infrastructures. Combined, the systems constitute a versatile tool for scalable eScience computations and demonstrate a functional approach for migration of local applications to distributed computing environments.

A. Stochastic Simulation of Reaction-Diffusion Kinetics

Modeling and simulation of reaction-diffusion processes are frequently employed in molecular systems biology to investigate networks of interacting macromolecules such as proteins. When modeling the dynamic behavior of cellular control systems on the molecular level, it is important to properly account for intrinsic stochasticity caused by small numbers of molecules. An important and frequently studied aspect of gene regulatory network function inside living cells is how they operate reliably despite large molecular fluctuations [16]. In such cases, deterministic phenomenological equations such as reaction-diffusion partial differential equations (PDEs) cannot be used to address the underlying biological questions.

Instead, stochasticity in the reaction-diffusion process can be accounted for by modeling it as a Markov process. Diffusion is modeled as jumps on a lattice, or mesh, and reactions occur according to pre-defined rules with rates given by the biochemical nature of the reactions and experimentally determined rate-constants. The state of the system is the discrete number of molecules of each biochemical species at each vertex in the mesh. Independent realizations of the process can be generated using kinetic Monte-Carlo (KMC) methodology. Unlike the deterministic PDE model, where a single solution is sufficient for one set of parameters and initial condition, a great many realizations of the stochastic process may be needed to analyze the properties of the biochemical network. For fine spatial discretization, the computational effort to generate even one such realization is high. Fortunately stochastic realizations are independent, and thus the overall problem is inherently task-parallel and maps well to distributed computing environments. This is a property shared with most computational workflows based on Monte Carlo methodology.

B. URDME

URDME [3] is a software package for stochastic reaction-diffusion simulation based on the described framework. It enables scientists to easily develop and execute simulations of models of biochemical networks and to analyze the resulting simulation data. URDME relies on the third party software COMSOL for geometry modeling, mesh generation, and pre- and postprocessing, and utilizes a Matlab interface to provide a familiar, interactive, and flexible environment for model

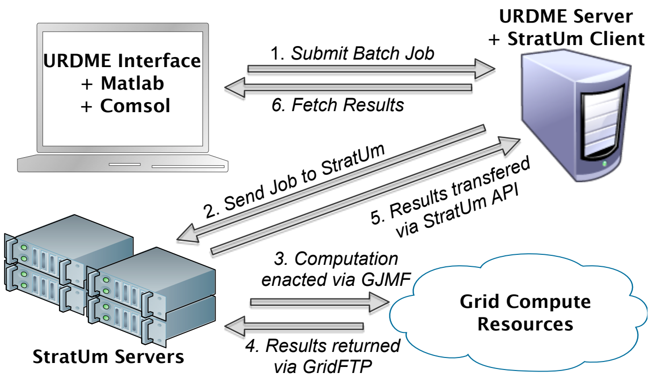

Fig. 2: Process flow for the integrated system. (1) URDME clients send batch computation requests to the URDME server. (2) The URDME server sends jobs (including data) to StratUm via the StratUm client APIs and RASP. (3) StratUm exacts computation on computational resources using GJMF. (4) Computation results are sent to storage servers via GridFTP. (5) StratUm data management service moves result data sets. (6) URDME clients asynchronously fetch result data sets.

development. The modular design of URDME allows easy extension of the core simulation algorithms of the package.

The current release of URDME is designed for an interactive usage pattern, thus to facilitate execution of URDME computations on distributed compute resources we have developed a new server-side component to the URDME framework that acts as a communication point between clients and StratUm. Figure 2 illustrates the process flow of a URDME job executed through StratUm in a distributed computation environment.

IV. Evaluation

To evaluate the system for use in production environments, we here demonstrate the use of the system to analyze a model of cell division in *E. Coli* using distributed resources from the Swedish Grid Initiative, SweGrid. The performance of the system is analyzed to characterize the system overhead.

A. Using URDME to conduct a parameter sweep on SweGrid

An important method for analyzing cellular reaction network models is to conduct parameter sweeps, in which parameters such as chemical reaction rate constants are varied within specified ranges. In such computations, model parameters undetermined by experiments can be fit by searching for areas in parameter space that reproduce observed or assumed behavior of the underlying biological system. Parameter sweeps conducting global sensitivity analysis can also reveal the robustness of the biochemical network to perturbations around the assumed parameter values of the rate constants. Often, parameter values derived from experiments are known with low accuracy. Sensitivity analysis can elucidate the precision with which the parameters must be known, and can contribute to qualitative understanding of the regulatory mechanisms modeled. This process requires generation of ensembles of simulations (for stochastic models) of the system for each point in parameter space. As simulation trajectories are independent, the process is well suited for distributed computation.

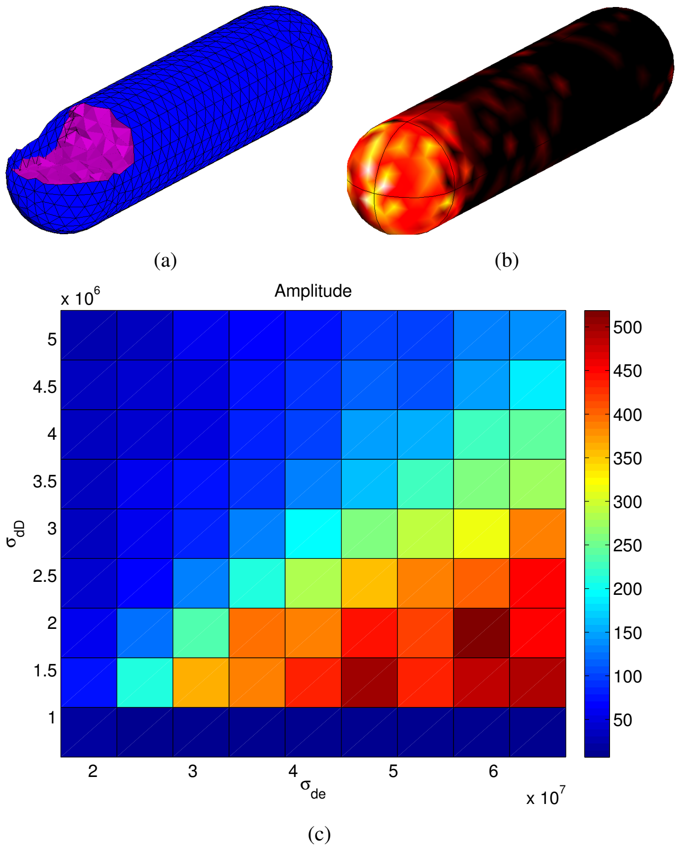

Fig. 3: The geometry of the bacterium is discretized on an unstructured mesh made up of tetrahedra in the interior and triangles on the boundary (a). A snapshot of the protein MinD on the membrane, localized to one of the poles in a stochastic URDME simulation (b). MinD oscillates from pole to pole. The amplitude of the oscillations vary with the rate constants in the model (c). Dark red regions indicate areas in parameter space where the amplitude is high and oscillations are consistent. We find the values of [8] to be near optimal.

To illustrate the use of the virtual infrastructure, we simulate a model of Min protein oscillations in the bacterium *E. Coli*. The Min system has been studied extensively in the literature using quantitative models, both in a deterministic setting and using a mesoscopic spatial stochastic framework [8], [6]. Here, we simulate the model from [8] using URDME. Fig. 3a shows the geometry and the mesh used to represent the bacterium. The boundary is represented by a surface mesh made up of triangles, and the interior volume is covered by non-overlapping tetrahedra. Diffusion jump constants on the unstructured mesh are obtained as detailed in [5]. For properly chosen parameters, a protein called MinD will oscillate regularly from pole to pole. These oscillations are believed to be a key component in the regulation of reliable and symmetric cell division.

To illustrate how the amplitude of the oscillations (i.e. the stability of the system) depends on two of the rate constants in the model, we conduct a two-dimensional parameter sweep in URDME, using StratUm to run simulations on SweGrid. Fig. 3c shows the outcome for a parameter grid consisting of 10 values for each of the rate constants σ_{dD} and σ_{de} [8]. To compute amplitude averages, we generate 10 independent realizations for each unique parameter combination. The total computation consists of 1000 individual tasks, each taking 5-10 minutes to execute (depending on parameter value).

B. Performance Evaluation

In this section we characterize the performance of the overall system when executed in a production grid environment.

1) Test Environment: The test environment used in the evaluation is comprised of a set of servers across Sweden. A 3.33 GHz 6 core Intel Core i7, 24 GB RAM server with a 100 Mbps network acting as URDME server host is deployed in Uppsala, and a dual quad-core (8 cores) AMD Opteron 1.8 GHz, 32 GB RAM server running a set of virtual machines is deployed in Umeå, Sweden. The Umeå virtual machines are configured with 1.8 GHz AMD Opteron 2 core CPUs, 4 GB RAM, and interconnected with a Gigabit Ethernet network.

The virtual machines are deployed with identical installations of Ubuntu Linux, StratUm, GJMF, and Globus Toolkit 4.0.5. In tests, they are the only load processes on the shared hardware server. An additional host is also running a (Globus) GridFTP server handling all file staging for computational jobs. The GJMF is configured to interconnect with SweGrid resources via the ARC [4] middleware, and communicates with the middleware via ARCLib v1. As GJMF overhead is independent of middleware overhead, and the purpose of the performance evaluation is to characterize StratUm and GJMF overhead contributions to total system overhead, a test setup using an older (Globus) middleware version is acceptable.

2) Performance Tests: The purpose of the experiments is to investigate individual overhead contributions to total system overhead and to relate the overhead imposed by StratUm and the GJMF to the functionality offered by the frameworks. Jobs are submitted from URDME to StratUm via RASP. StratUm utilizes the GJMF to submit and control jobs at SweGrid. Experiment data are staged via StratUm to a GridFTP server that manages file staging to and from SweGrid resources.

Synchronized clock timestamps are used to determine the amount of time jobs spend in different systems, which forms the basis for characterization of system behavior. In distributed system environments, job executions suffer overhead for file transfers (staging) to and from computational resources, and most likely also (due to resource contention) for scheduling and queuing (wait) time on resources. Tests utilize a model for execution of a single job that categorizes time spent as

1) StratUm stage-in: client to storage file transfers.
2) overhead imposed by use of StratUm.
3) overhead imposed by use of GJMF.
4) overhead imposed by failed jobs.
5) stage-in: storage to resource file transfers.
6) wait time: resource scheduling and queuing.
7) job execution: execution of jobs at resources.
8) stage-out: resource to storage file transfers.
9) StratUm stage-out: storage to client file transfers.

In reality, resource stage-in and resource wait time are typically overlapped, as they may be performed in parallel in resource scheduling environments. It should be noted that utilization of StratUm for file staging to and from storage servers is an optional abstraction mechanism for clients that do not wish to interface with storage servers directly. In tests we have not included information about file staging from storage servers to clients, as the chosen experiment produces data that are incrementally post-processed by clients.

3) Analysis: As the integration system is made up of multiple hierarchical systems that perform tasks in parallel, analysis of job behavior is non-trivial. Overhead imposed by using distributed computational resources is substantial, and can for analysis be classified by overhead source. Figure 4 illustrates the average relative distribution of overhead for computational jobs run in the experiment. As can be seen (in the upper pie chart), a majority of the overhead imposed by using distributed computational resources is spent performing file staging or waiting for execution on resources. As this overhead would be imposed even if the underlying grid middleware would be used directly, it is considered part of the middleware overhead and not part of the cost for using StratUm and GJMF. Further analysis of the overhead contributed by StratUm and GJMF (illustrated in the lower pie chart of Figure 4) reveals that the majority of the StratUm overhead in these experiments is constituted by the optional file stage-in activity. Note that overhead from failed jobs is accounted for as GJMF overhead, as the GJMF automates rescheduling of failed jobs.

As illustrated in the histogram of Figure 4, most of the overhead contributed by StratUm and GJMF is incurred early in the job execution process. Job executions, file stagings, etc. are monitored by the systems, but monitoring overhead is masked by parallelism with other system tasks. In the experiment, an average of less than 15 and 13 seconds of overhead per job is imposed by StratUm and GJMF. Even with the short-lived computational jobs of this setting, this constitutes less than 1% of the total execution time.

The value of high abstraction levels and automation of computation management is well illustrated by the failure rates of distributed computing environments. In the experiment, 0.5% of the jobs failed and were automatically rerun by the GJMF. The time (roughly 4.5 hours) these failed jobs spent performing file staging, execution, etc. can be considered lost experiment time. Identifying and recovering from these failures could, however, potentially delay a large-scale experiment substantially longer than the lost experiment time. This illustrates that for large experiments, failures constitute non-negligible overhead, and abstractive mechanisms for automated management of computations can be very valuable.

In summary, the behavior of computational jobs in an architecture such as the proposed integration system will largely depend on characteristics of the underlying computational infrastructure used. The average overhead for managing computations using tools designed with the approach explored here

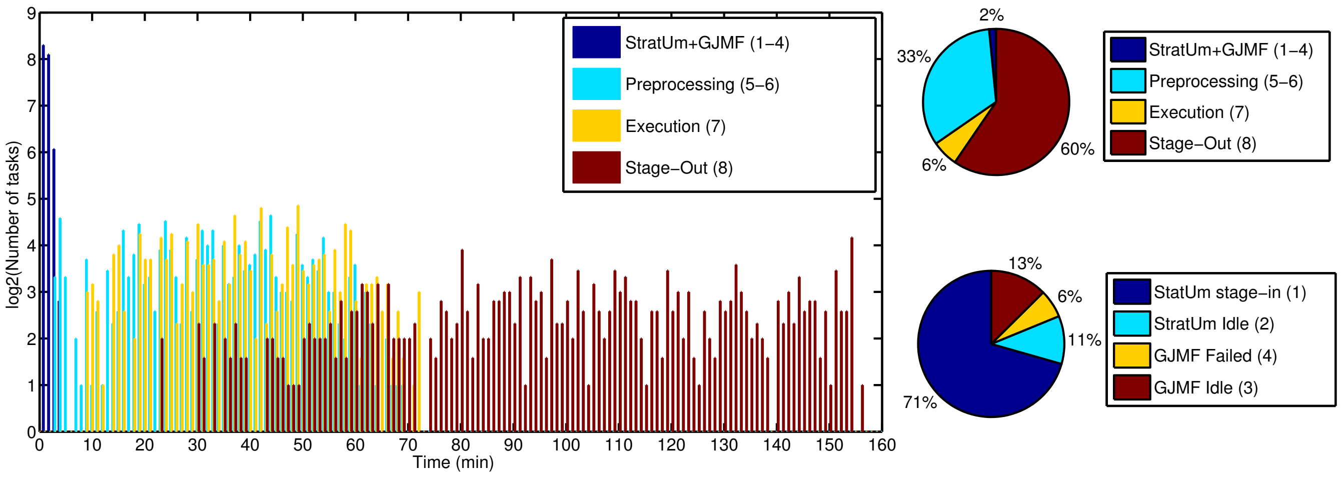

Fig. 4: Relative distributions of time components for job executions. The histogram shows the number of tasks in the respective stages of the execution process as a function of time. Pie charts of average relative distributions of time components for individual jobs (upper) and relative distribution of internal StratUm and GJMF overhead components (lower).

is low enough to motivate use of the systems. The techniques used here are applicable to most types of computational environments. As the overhead is low, the performance of the systems will largely be characterized by the performance of the underlying computational infrastructure.

V. Discussion

From the application developer perspective, the StratUm toolkit greatly simplifies migration from local resources to distributed resources in this case study. Development of specialized StratUm clients, that compartmentalize an integration bridge between the GJMF and URDME, allows extension and development of the existing URDME infrastructure without modification of the conceptual model of the application. The integration clients build on the StratUm native client APIs and are customized to interface with the application as seamlessly as possible, e.g., through organizing and managing computational tasks in groups as inferred from the URDME computational structures. Communication between the clients and the pre-existing application software is managed through a small set of Bash and Perl scripts as part of a new server-side component of URDME. Importantly, by allowing for this design of the overall system, no alteration of the pre-existing code base is required – all necessary new components are implemented in a dedicated layer of code maintained and released separately from the main URDME package.

From a modeling point of view, the stochastic models in this case study can provide more accurate descriptions of the biochemical systems than the more traditional, deterministic models based on PDEs. From a computational point of view, the nature of computations is fundamentally different between these two models, changing from one single, large synchronized computation for the PDE model to large numbers of task parallel computations for the stochastic model. Computational applications such as URDME can however generate computations that vary in behavior based on the nature of the task. In the URDME case, the nature of the molecular network under study, as well as the mesh resolution, affects the average number of timesteps taken by the KMC method for a given time-horizon of the simulation. The time-horizon in turn depends on the biological question addressed: it may for example be necessary to observe the system on the timescales of several minutes to characterize the reliability of the oscillations, while discrete events in the simulations may occur on nanosecond timescales for fine meshes.

From a practical standpoint, this means that for the same application (URDME) computational tasks will vary greatly in, e.g., number and compute-to-data transfer ratios. In the evaluation, we conducted a parameter sweep using a rather coarse mesh, and the compute times of tasks (realizations) are short ($\sim$5-10 minutes per core on a modern workstation). This results, as illustrated in Figure 4, in a situation where the average execution time constitutes only a small part (6%) of the overall time required for jobs to complete. In larger scientific inquiries, these computations could constitute the first steps in a larger experiment where a parameter space is explored by initially conducting simulations with low spatial resolution and for many different parameters, followed by (when potentially interesting regions of parameter space are identified) local refinement of the parameter space. In such cases, computations will transition from data-intensive to compute-intensive, which will require the underlying computational infrastructure to accommodate the dynamic nature of this process.

To increase application end-user productivity, it is desirable to be able to modify system resource utilization behavior to, e.g., maximize throughput or minimize asynchrony between groups of tasks with dependencies in a larger workflow. To this end, the StratUm client APIs are designed to provide flexibility for customizations with respect to *different applica-*

tions, but also to open up for optimizations and customization for *different use cases of the same application*. In the case study, the application (URDME) is able to predict the behavior of computational tasks for different use-cases and biological models, and the integration model between the application and the computation enactment system (StratUm) is therefore designed to give the flexibility in resource utilization required to maximize computational throughput.

In addition to computation, storage of, and access to, resulting simulation data for post-processing and analysis is an important aspect of the overall system workflow. As the detailed nature of the post-processing requirements are rarely known in advance for a given biochemical model, it is very hard for the application to anticipate and provide routines that will serve all users. The local version of the URDME software addresses this issue by providing access to simulation routines and result data in the Matlab scripting environment. This flexibility needs to be preserved even when simulations and data are conducted and stored on distributed resources. Again, the design of StratUm makes possible a flexible choice of data-transfer protocols on the application side. To facilitate convenient integration of data storage and analysis, the StratUm integration bridge clients encompass functionality for abstracting data management.

VI. Conclusion

In this paper we discuss migration of computational applications to dedicated and distributed resource environments and present StratUm, an extensible toolkit for facilitating the computational needs of eScience applications. We extend on earlier work and build on a pattern of separation of computation design and execution of computations in distributed resource environments. The approach focuses on development of generic infrastructure components that can be reused between projects, coupled with integration tools that are customized to computational applications. The aim of the toolkit is to reduce computation management complexity and facilitate migration of applications to dedicated resource environments without altering the way end-users design computations.

To illustrate the approach, we present StratUm in the context of a case study where we integrate a computational systems biology application with a grid job management framework. The resulting system constitutes a powerful tool for mesoscopic spatial stochastic simulation and is evaluated in a computational experiment run on the Swedish national grid. Evaluation results indicate that StratUm contributes limited overhead to distributed computation scenarios, and that computation management complexity is well abstracted by the toolkit in the case study. Integration findings and migration methodology are discussed throughout the paper.

VII. Acknowledgements

The authors acknowledge Mikael Öhman, Sebastian Gröhn, and Anders Häggström for work related to the project, and the anonymous reviewers for valuable feedback and interesting discussions. This work is done in collaboration with the High Performance Computing Center North (HPC2N) and is funded by the Swedish National Infrastructure for Computing (SNIC), the Swedish Government's strategic research project eSSENCE, the Swedish Royal Academy of Sciences, and U.S NSF Grant DMS-1001012, U.S. DOE Award DE-FG02-04ER25621, U.S. NSF IGERT DGE-02-21715, and Institute for Collaborative Biotechnologies Grant DAAD19-03-D-0004 from the U.S. Army Research Office.

References

[1] C. Bunch, N. Chohan, C. Krintz, and K. Shams. Neptune: a domain specific language for deploying hpc software on cloud platforms. In *Proceedings of the 2nd international workshop on Scientific cloud computing*, ScienceCloud '11, pages 59–68, New York, NY, USA, 2011. ACM.

[2] H. Casanova, F. Berman, T. Bartol, E. Gokcay, T. Sejnowski, A. Birnbaum, J. Dongarra, M. Miller, M. Ellisman, M. Faerman, G. Obertelli, R. Wolski, S. Pomerantz, and J. Stiles. The virtual instrument: Support for grid-enabled mcell simulations. *International Journal of High Performance Computing Applications*, 18(1):3–17, 2004.

[3] B. Drawert, S. Engblom, and A. Hellander. URDME 1.1: User's manual. Technical Report 2011-003, Department of Information Technology, Division of Scientific Computing, Uppsala University, 2011.

[4] M. Ellert and et.al. Advanced resource connector middleware for lightweight computational Grids. *Future Generation Computer Systems. The International Journal of Grid Computing: Theory, Methods and Applications*, 27(2):219–240, 2007.

[5] S. Engblom, L. Ferm, A. Hellander, and P. Lötstedt. Simulation of stochastic reaction–diffusion processes on unstructured meshes. *SIAM J. Sci. Comput.*, 31(3):1774–1797, 2009.

[6] D. Fange and J. Elf. Noise induced Min phenotypes in *E. coli*. *PLoS Comput. Biol.*, 2(6):e80, 2006.

[7] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In H. Jin, D. Reed, and W. Jiang, editors, *IFIP International Conference on Network and Parallel Computing, LNCS 3779*, pages 2–13. Springer-Verlag, 2005.

[8] K. C. Huang, Y. Meir, and N. S. Wingreen. Dynamic structures in *escherichia coli*: Spontaneous formation of MinE and MinD polar zones. *Proc. Natl. Acad. Sci. USA*, 100(22):12724–12728, 2003.

[9] H. Kaiser, A. Merzky, S. Hirmer, G. Allen, and E. Seidel. The saga c++ reference implementation: a milestone toward new high-level grid applications. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.

[10] W. Lee, A. S. McGough, and J. Darlington. Performance evaluation of the GridSAM job submission and monitoring system. In *UK e-Science All Hands Meeting*, pages 915–922, 2005.

[11] P-O. Östberg and E. Elmroth. Mediation of Service Overhead in Service-Oriented Grid Architectures. In S. Jha, N.g. Felde, R. Buyya, and G. Fedak, editors, *Proceedings of Grid 2011: 12th IEEE/ACM International Conference on Grid Computing*, pages 9–18, 2011.

[12] P-O. Östberg and E. Elmroth. GJMF - A Composable Service-Oriented Grid Job Management Framework. Preprint available at http://www.cs.umu.se/ds, submitted, 2010.

[13] P-O. Östberg, A. Hellander, B. Drawert, E. Elmroth, S. Holmgren, and L. Petzold. Abstractions For Scaling eScience Applications to Distributed Computing Environments. In *Proceedings of BIOINFORMATICS 2012, International Conference on Bioinformatics Models, Methods, and Algorithms*, pages 290–294, 2012.

[14] I. Raicu, I.T. Foster, and Y. Zhao. Many-task computing for grids and supercomputers. In *Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS) 2008.*, pages 1–11, 2008.

[15] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde. Falkon: a Fast and Light-weight tasK executiON framework. In *Proceedings of IEEE/ACM Supercomputing 07*, 2007.

[16] P. S. Swain. Efficient attenuation of stochasticity in gene expression through post-transcriptional control. *J. Mol. Biol.*, 344(4):965 – 976, 2004.

[17] M. Welsh, D. Culler, and E. Brewer. SEDA: An architecture for well-connected scalable internet services. *Operating System Review*, 35(5):230–243, 2001.