Accelerating Parameter Sweep Workflows by Utilizing Ad-hoc Network Computing Resources: an Ecological Example

Jianwu Wang[1], Ilkay Altintas[1], Parviez R. Hosseini[2], Derik Barseghian[3], Daniel Crawl[1], Chad Berkley[3], Matthew B. Jones[3]

[1] *San Diego Supercomputer Center, UCSD, U.S.A.*
{jianwu, altintas, crawl}@sdsc.edu

[2] *The Consortium for Conservation Medicine, Wildlife Trust, U.S.A.*
hosseini@princeton.edu

[3] *National Center for Ecological Analysis and Synthesis, UCSB, U.S.A.*
{barseghian, berkley, jones}@nceas.ucsb.edu

Abstract

Making use of distributed execution within scientific workflows is a growing and promising methodology to achieve better execution performance. We have implemented a distributed execution framework in the Kepler scientific workflow environment, called Master-Slave Distribution, to distribute sub-workflows to a common distributed environment, namely ad-hoc network computing resources. For a typical parameter sweep workflow, this architecture can realize concurrent independent sub-workflow executions with minimal user configuration, allowing large gains in productivity with little of the typical overhead associated with learning distributed computing systems. We explain details of the Master-Slave architecture and demonstrate its usability and time efficiency by a use case in the theoretical ecology domain. We also discuss the capabilities of this architecture under different computational domains in Kepler.

1. Introduction

Scientific workflow management systems, e.g., Kepler [1], Taverna [2], Triana [3], Pegasus [4], ASKALON [5] and SWIFT [6], have demonstrated their ability to help domain scientists solve scientific problems by synthesizing different data and computing resources. Scientific workflows can operate at different levels of granularity, from low-level workflows that explicitly move data around and monitor remote jobs, to high-level "conceptual workflows" that interlink complex, domain-specific data analysis steps.

Many scientific computing problems have linear or greater time complexity, with execution times ranging from milliseconds to hours or even days. For small parameter configurations that result in few runs, a single notebook computer can handle workflow execution at times. However, for large parameter configurations that involve many permutations or intensive computations, the execution tasks may require other computing resources to accelerate execution. Even though these scientific problems would benefit from increased computational resources, the configuration complexity associated with most distributed systems in use today effectively prevents scientists from adopting and using them. For a distributed system to be effective, it must both provide access to the necessary resources and be easily configurable by practicing scientists who are not familiar with distributed computing software. Thus, the problem we address is how to smoothly transition between execution environments; a workflow should function both when distributed computing resources are available and when they are not, and a user should be able to easily leverage distributing computing resources with little knowledge of the underlying distributed system.

There are many kinds of sophisticated distributed environments that can be utilized for workflow execution, such as Cluster, Grid and Cloud computing [4][5][6][7][8][9]. A simpler approach is an ad-hoc network comprised of independent computers. We utilize such a network to accelerate workflow execution, implementing an architecture in Kepler called *Master-Slave*. In this paper, we will discuss the application of this architecture for parameter sweep applications and workflows [7][10], which are common in many scientific domains and involve independent multiple execution, i.e., "embarrassingly parallel problems". Compute-intensive tasks in these workflows can be distributed among ad-hoc network nodes and executed in parallel.

In Section 2, we describe a real-world parameter sweep workflow use case in the theoretical ecology domain that we use to validate system capability and usability. Section 3 describes the background of our architecture. Our approach and its results for the use case are explained in Sections 4 and 5. We compare our work with related work in Section 6. Finally, we conclude and explain future work in Section 7.

2. Theoretical Ecology Use Case

Our domain example is a spatial stochastic birth-death process [11][12] that simulates the dynamics of *Mycoplasma gallisepticum* in House Finches (*Carpodacus mexicanus*) [13]. The use case needs to be run over a broad range of parameters, such as the birth rate of finches (parameter *b* in Table 1) and the death rate of finches (parameter *d* in Table 1). These parameters are analogous to those in [13]. Additionally, we can test the scalability of the problem, as its spatio-temporal aspect can be solved quickly for simulations with short temporal lengths (parameter *E*, end time of simulation), and small spatial areas (parameters *X* and *Y*, the breadth and width of the simulated spatial domain), but the process can consume a large amount of computation time for longer temporal solutions of larger simulated spatial domains. The simulation code is written in GNU C++, and involves file reads, relatively complex mathematical operations, and delayed functor calls. The compiled executable, and thus the simulation itself, is called *simhofi*. The execution results were visualized using the R statistical system, as no graphical output was coded into the simulation program itself.

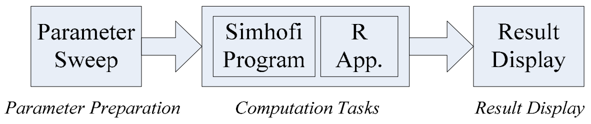


Figure 1. Conceptual workflow for ecology use case.

The conceptual workflow for this use case has three main parts (Figure 1): parameter preparation, computation tasks, and result display. The workflow was initially designed to be executed only on a single computer, and it works well with small parameter ranges. The execution time of each iteration of the program is proportional to $X * Y * E$. So when the ecologist needs simulation results for longer temporal solutions of larger simulated spatial domains, the execution time of the workflow increases rapidly, e.g., a single iteration of the simulation can take 5 hours for X=32, Y=32, E=30, and a given parameter sweep might involve tens or hundreds of iterations.

This use case is typical within the Kepler community. Implementations of such workflows using Kepler's Master-Slave Distribution Framework reduce execution time by utilizing distributed computing resources with minimal changes to the workflows compared to the non-distributed case. The main characteristics of the use case are as follows:

- **Parameter Sweep**: Parameter sweep applications and workflows [7][10] are common in many scientific domains. Compute-intensive tasks in one workflow need to be executed many times with different parameter permutations. Each execution could take a long time and is independent of the others.
- **Smooth Transition of Computation Environments**: A common empirical method for workflow design is to test workflow logic with small parameters locally at first, and then change the computation environments of the workflow to distributed environments to accelerate workflow execution with large parameters. A major goal of our design is to make this transition as easy as possible for workflow users, while minimizing workflow modifications.
- **Partial Workflow Distribution:** Users may still need some parts of the workflow to run locally. It may be for easy interaction, such as parameter preparation and results display, or privacy, such as some tasks that need to be executed on local, private data.
- **Provenance Collection:** Users may need to track workflow output data generated by domain specific programs according to given parameter ranges. Provenance collection should work in both local and distributed environments.

3. Background

3.1. Kepler

The Kepler project[1] aims to produce an open-source scientific workflow system that allows scientists to design and efficiently execute scientific workflows. Since 2003, Kepler has been used as a workflow system within over 20 diverse projects and multiple disciplines.

Inherited from Ptolemy II[2], Kepler adopts the *actor-oriented modeling* [1] paradigm for scientific workflow design and execution. Each actor is designed to perform a specific independent task which can be implemented as "atomic" actors and "composite" actors. Composite actors are collections or sets of atomic actors bundled together to perform more complex operations. All these actors inherit the same interfaces, such as *prefire()*, *fire()* and *postfire()*, which tell the actor what to do at various times during workflow execution.

Another unique property inherited from Ptolemy II is that the order of execution of actors in the workflow is specified by an independent entity called the *director*. The director defines how actors are executed and how they communicate with each other. The execution model defined by the director is called the *Model of Computation* [1]. Since the director is decoupled from the workflow structure, the user can easily change the computation model by replacing the director using the Kepler graphical user interface. As a consequence, a single actor can execute both in a sequential manner, e.g., using the Synchronous Data Flow (SDF) director or in a parallel manner e.g., using the Process Network (PN) director.

3.2. Master-Slave Architecture

We present a high-level distributed execution framework for scientific workflows in [14]. In this distributed framework, illustrated in Figure 2, each computing node runs an instance of the workflow execution engine and is assigned one or more roles. Workflow execution is initiated by a *Master* node that performs overall coordination of an arbitrary number of *Slave* nodes that execute sub-workflow tasks. Additionally, a *Registration Center* and a *Provenance Manager* broker Slave execution capability and data generated during workflow execution, respectively.

[1] http://www.kepler-project.org/
[2] http://ptolemy.eecs.berkeley.edu/ptolemyII/

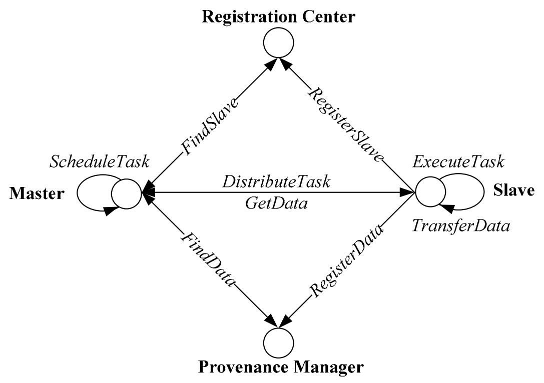


Figure 2: Master-slave architecture for workflow distributed execution.

4. Approach

Our previous paper [14] discussed this architecture from a general and high-level perspective. In this section, we only address implementation details that are unique to Kepler.

4.1. Distributed Composite Actor

A distributed composite actor was designed, called *DistributedCompositeActor*, to act as the role of Master. Each data received by the DistributedCompositeActor is distributed to a Slave node, and the result data is returned to the Master node by the Slave node. This actor has different behavior with different computation models [1]. We will discuss its behavior with two typical computation models: Synchronous Dataflow (SDF) and Process Networks (PN). This behavior is illustrated in Figure 3.

In the SDF model, an actor consumes and produces a fixed number of input and output data per firing. When executed in this model, DistributedCompositeActor can only distribute the current input data to one Slave per firing. The Slaves will be executed synchronously since DistributedCompositeActor is only allowed to schedule and execute one Slave at a time. It is more suitable for dependent task execution, e.g., Markov Chain Monte Carlo, where the inputs of the next sub-workflow are dependent on the outputs of the former one.

In the PN model, a workflow is driven by data availability: all actors in a workflow execute in their own active threads and an actor continues to execute until no input data are available. In this model, DistributedCompositeActor schedules and executes all available Slaves in one firing based on availability of input data. Concretely, DistributedCompositeActor monitors the execution of each Slave, and once one Slave finishes its current execution, DistributedCompositeActor will send the next input data to this Slave for processing. The Slaves therefore execute concurrently with different input data. Additional Slaves increase parallel computation and faster Slaves will execute more frequently. Because some nodes might calculate results in shorter time periods than others, output data may not be returned in the same order in which they were sent. Thus, this model is suitable for independent, multiple task execution. Parameter sweep workflow can utilize the Process Network model to improve execution efficiency by distributing tasks in parallel to remote Slave nodes.

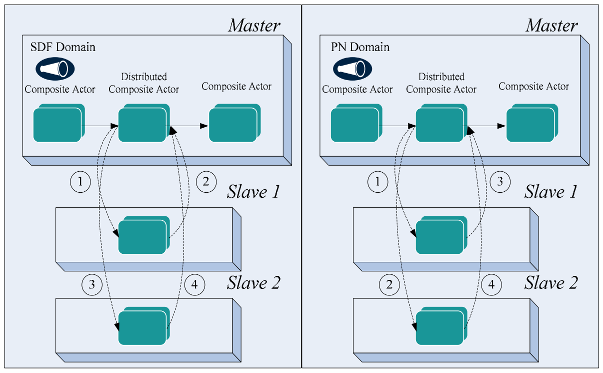


Figure 3: Behavior of DistributedCompositeActor with different computation models. The order in which data is sent and returned depends on the computational model used.

Since DistributedCompositeActor inherits the same interfaces of other actors, it is very easy to switch between the regular composite actor and DistributedCompositeActor. For workflow users, this actor is very similar to the regular composite actor except a few additional configuration parameters for the Slaves, such as the Slave host URL. From a workflow specification perspective, the only change is the actor class name and some attributes. All other workflow specification details, such as inner actors and links between actors, are still the same. From the user's perspective, a user can switch between a regular composite actor and DistributedCompositeActor just by right clicking on the composite actor.

For the Slave role of our Master-Slave architecture, we implemented a Java Remote Method Invocation (RMI) service that wraps the Kepler execution engine and that communicates with the Master through the underlying RMI infrastructure. For each Slave node, a Slave package and the domain specific programs to be invoked, such as the simhofi and R programs in the above use case, must be deployed on the Slave node beforehand. During the initial phase of the workflow, DistributedCompositeActor will transfer its specification to each Slave. Then the Slaves are ready for receiving input data and execution. The output data will be transferred back to the Master node once the current iteration is finished.

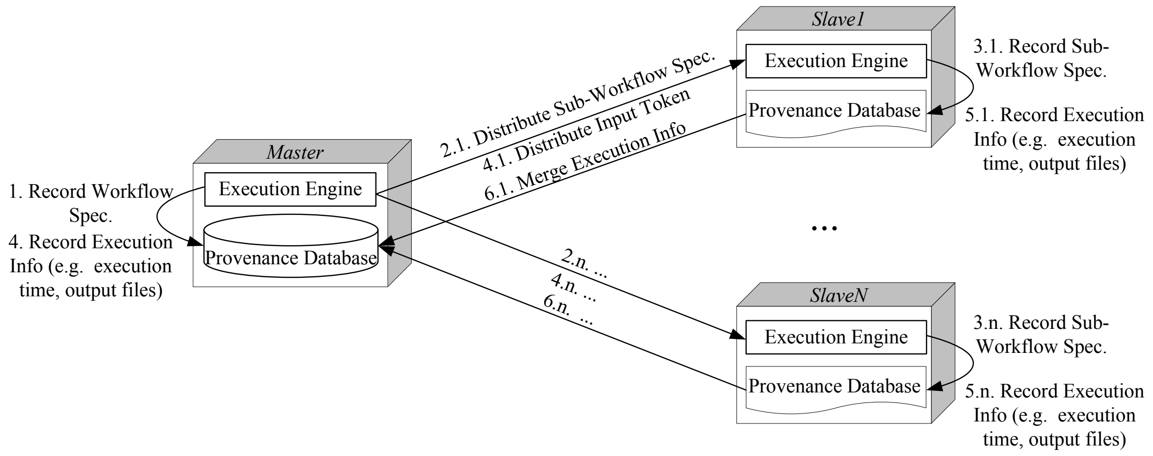
To manage available Slave information, a centralized Web service, called the *Registration Service,* is provided. When a Slave RMI service is started, it invokes the Registration Service to register this node as a Slave. This allows the Master node to get a listing of all available Slave nodes by querying the Registration Service. We are extending the Registration Service to support more detailed Slave capability metadata and improved Slave selection [14].

In addition to the Registration Service, we also provide the *Authentication Service*, a web service for user authentication. Users need to login to see the available Slaves and their credentials are needed for Slave operations, such as *start*, *stop*, *getStatus*. This service is currently implemented using LDAP, but could also be implemented using other authentication systems by implementing the service interface.

4.2. Provenance Collection

The Kepler Provenance Framework collects information about workflow structure and executions [15]. We have implemented several new provenance-related features for this use case. Before the workflow executes, it saves a copy of the workflow. Additionally, the contents of files generated by actors during workflow execution are now stored. In this use case, the generated data files and R visualizations are stored in the provenance database, along with the workflow (including all input parameter values) that created them. In general, for each possible combination in a parameter sweep, one or more domain specific programs will generate output files; our provenance features make it easier for users to track data files for large parameter sweeps.

If a workflow is executed within the Master-Slave architecture, the related distributed provenance information will be automatically recorded, such as Slave node information. Since Slave nodes used in previous workflow executions may not always be accessible, one challenge for a distributed provenance framework is where to store provenance information generated on Slaves. To deal with this issue, our provenance framework can be configured to support centralized or decentralized provenance information recording. In the former case, each Slave writes provenance information directly to a centralized database. Whereas in the latter, as illustrated in Figure 4, each Slave first writes to a local file-based database, and the provenance information is merged into a centralized database upon workflow completion.



Slave1
Execution Engine
Provenance Database
3.1. Record Sub-Workflow Spec.
5.1. Record Execution Info (e.g. execution time, output files)
2.1. Distribute Sub-Workflow Spec.
4.1. Distribute Input Token
6.1. Merge Execution Info
Master
Execution Engine
Provenance Database
1. Record Workflow Spec.
4. Record Execution Info (e.g. execution time, output files)
...
2.n. ...
4.n. ...
6.n. ...
SlaveN
Execution Engine
Provenance Database
3.n. Record Sub-Workflow Spec.
5.n. Record Execution Info (e.g. execution time, output files)

Figure 4: Interaction sequences for distributed provenance.

5. Results

5.1. Workflow

The Kepler workflow for the above use case is shown in Figure 5. The top-level workflow is shown in the lower left of the figure. Its first part is for parameter preparation. House Finch Death Rate (*d*) and House Finch Birth Rate (*b*) are two parameters when combined generate 8 parameter combinations. In the composite actor *Parameter Sweep*, these two parameters are permuted and generate the corresponding input data for the next actor. Actor *Distributed Simhofi*, which contains the main computation tasks of the whole workflow, is a DistributedCompositeActor that will be distributed to Slaves, and its inner structure is shown in the upper right of Figure 5. It uses the *External Execution* actor to invoke the simhofi program with its parameters, and then invokes the *R Expression* actor to generate visualizations of the simhofi execution result. The last two actors of the top level workflow, *R Output* and *ImageJ*, display the results. The visual simulation result (bottom right of Figure 5) will be shown through the ImageJ actor and the concrete text information will be shown at the R Output actor. Besides the main workflow structure, the blue rectangle in the top level workflow, called *Provenance Recorder*, specifies that provenance information will be recorded.

5.2. Usability

We hide many technical details from users in the workflow, including data transfer and Slave management details. Users use the DistributedCompositeActor just like the common composite actor. This workflow also represents a common design structure for parameter sweep applications in Kepler workflows, which can be easily adapted to other parameter sweep applications.

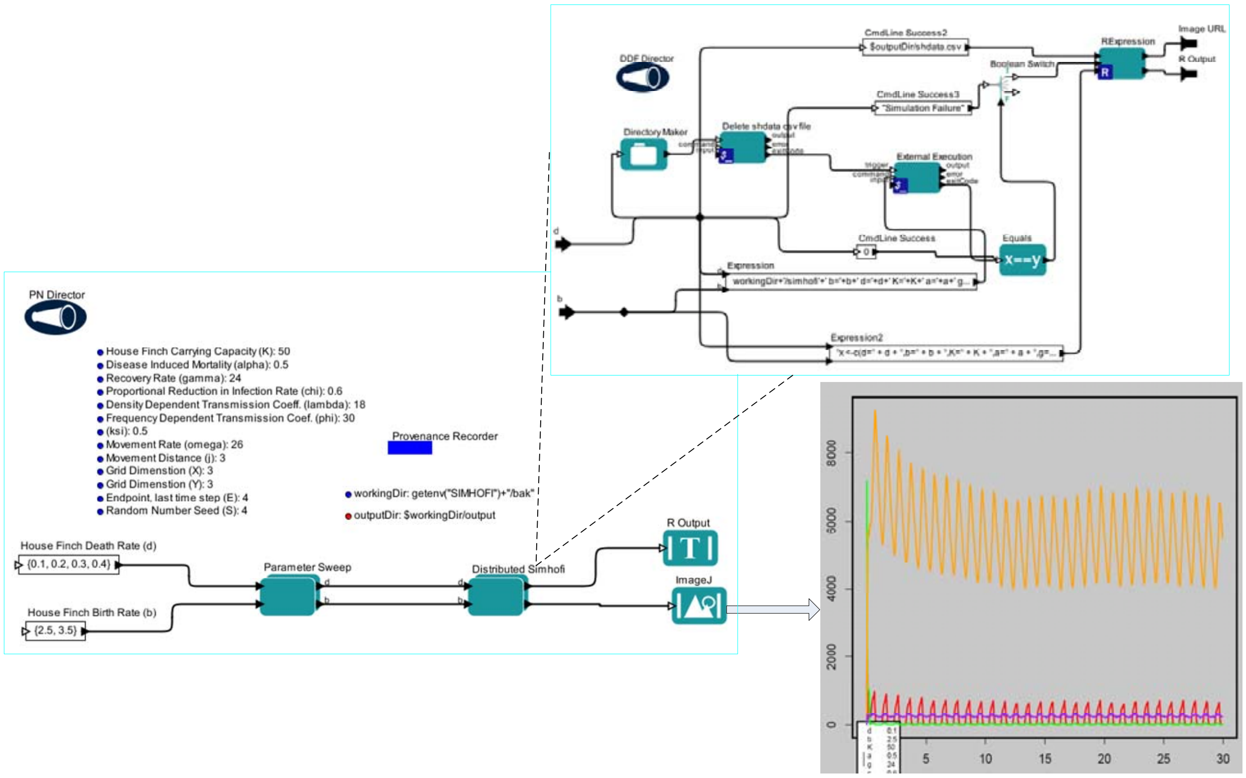
As illustrated in Figure 6, if a user wants to change her existing composite actor to the distributed one, she just needs to right click and choose ‘Distribute This Actor’ menu item. A dialogue will then be shown for Slave selection. After this configuration, the actor is converted to a distributed one and ready to execute on the chosen Slave node(s). The new workflow is identical except for the actor class name and a few attribute values.

5.3. Experiment

To compare the time efficiency of Master-Slave and its behavior with different computation models, we ran the workflow locally and with the Master-Slave architecture, both with the SDF and PN directors. We also recorded the workflow execution time with different parameter configurations to examine when the use of the Master-Slave architecture is beneficial.

In our experiment, we used one computer for local execution, and used this computer and one additional for Master-Slave execution. Since one computer can act as both Master and Slave node, we use the two computers as Slaves. The concrete experiment data is shown in Table 1 and Figure 7.

The experimental data shows that the Master-Slave architecture improves execution efficiency for large data sets and multiple independent program executions, whereas its overhead reduces efficiency for short execution runs. The PN director is suitable for the parameter sweep workflow and the experimental results show the PN director for Master-Slave architecture is more efficient than SDF director.



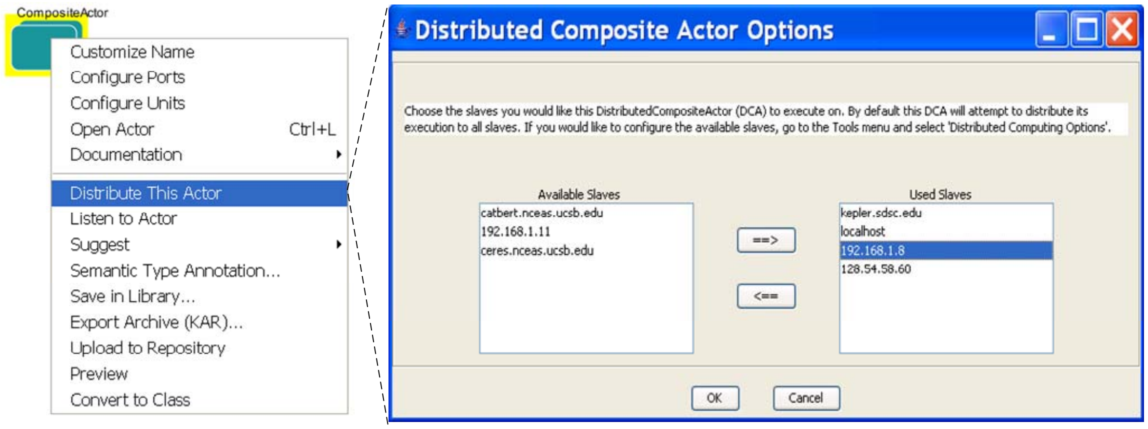
- House Finch Carrying Capacity (K): 50
- Disease Induced Mortality (alpha): 0.5
- Recovery Rate (gamma): 24
- Proportional Reduction in Infection Rate (chi): 0.6
- Density Dependent Transmission Coeff. (lambda): 18
- Frequency Dependent Transmission Coef. (phi): 30
- (ksi): 0.5
- Movement Rate (omega): 26
- Movement Distance (j): 3
- Grid Dimenstion (X): 3
- Grid Dimenstion (Y): 3
- Endpoint, last time step (E): 4
- Random Number Seed (S): 4

- workingDir: getenv("SIMHOFI")+"/bak"
- outputDir: $workingDir/output

House Finch Death Rate (d): {0.1, 0.2, 0.3, 0.4}

House Finch Birth Rate (b): {2.5, 3.5}

Figure 5: Distributed workflow for ecology use case, showing the overall workflow in the lower left and the portion of the workflow that is distributed to Slave nodes in the upper right.



Distributed Composite Actor Options

Choose the slaves you would like this DistributedCompositeActor (DCA) to execute on. By default this DCA will attempt to distribute its execution to all slaves. If you would like to configure the available slaves, go to the Tools menu and select 'Distributed Computing Options'.

Available Slaves	Used Slaves
catbert.nceas.ucsb.edu	kepler.sdsc.edu
192.168.1.11	localhost
ceres.nceas.ucsb.edu	192.168.1.8
	128.54.58.60

Figure 6: Interaction for execution environment transition.

Table 1: Comparative execution times for local and distributed execution of the ecology use case.

Parameters	Execution Time (minutes)			
	SDF locally	**PN locally**	**SDF Master-Slave**	**PN Master-Slave**
b=<0.1, 0.2, 0.3, 0.4>, *d*=<2.5, 3.5>, *X*=3, *Y*=3, *E*=4	0.39	0.35	0.60	0.52
b=<0.1, 0.2, 0.3, 0.4>, *d*=<2.5, 3.5>, *X*=8, *Y*=8, *E*=30	32.21	32.24	19.05	9.38
b=<0.1, 0.2, 0.3, 0.4>, *d*=<2.5, 3.5>, *X*=16, *Y*=16, *E*=30	502.2	510	309	147
Testbed Constitution				
	OS	Memory	CPU	
Notebook	Window XP	2 GB	2.00 GHz Duo Core	
Desktop	Mac OS X	2 GB	2.80 GHz Duo Core	

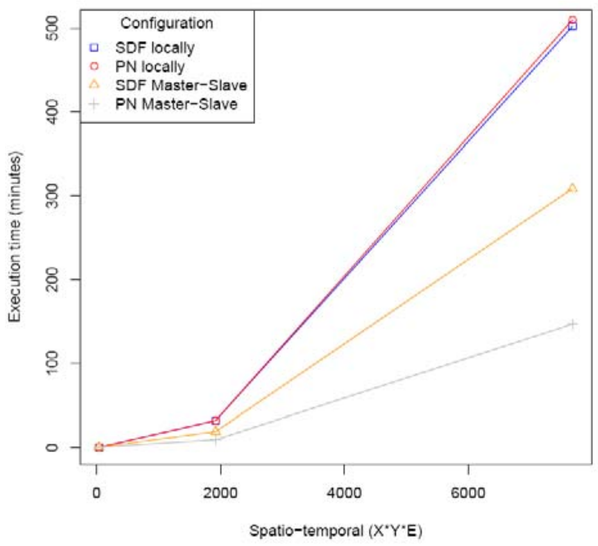


Figure 7: Experiment data illustration for the ecology use case.

6. Related Work

There have been several workflow systems [4][5][6][7][8][9] that can utilize distributed environments, such as Cluster, Grid and Cloud computing, to accelerate workflow execution. However, most of them need job management system support, and this may be too heavyweight or expensive for some scientific computing problems. In addition, these workflows themselves are dependent on the distributed environments to some extent, making them more difficult to configure. In these systems, users may need to know, e.g., a specific job description specification syntax to construct the workflow. Further, converting locally executable workflows to ones that execute in most distributed environment systems is not trivial. Most of these do not allow partial workflow distribution rather than submitting the whole workflow to distributed systems. Comparing the above work, our Master-Slave architecture focuses on the lightweight and usability objectives. With the RMI infrastructure, each computer can easily join and exit the architecture; no additional distributed resource management system, such as, Condor [16] and Globus [17], is needed. Additionally, users can easily transition sub-workflows from local execution to ad-hoc network based distributed execution.

Triana [3] supports a function similar to our Master-Slave architecture by distributing sub-workflows to remote services. It provides two kinds of distributed execution topologies: parallel task computation and pipelined connectivity with direct data transfer between tasks, which is similar to our DistributedCompositeActor but lacks the distributed provenance and security frameworks that we present for Kepler.

7. Conclusions and Future Work

Parameter sweep applications in which the programs can be executed independently for each parameter combination are common in many scientific domains. A simple and common distributed environment is an ad-hoc network comprised of independent computers. We discuss the application of our Master-Slave architecture which utilizes ad-hoc network computing resources to accelerate these kinds of workflows. For many scientists who are unfamiliar with Cluster or Grid computing, the type of ad-hoc network that we demonstrate can be extremely beneficial. It allows scientists to utilize the dozens of computers that they may have available in their local network with minimal overhead and little or no knowledge of complex distributed computing frameworks. All they need is Kepler installed on each of the nodes. By analyzing the performance of this architecture under different computational domains in Kepler, we show that there are benefits to using the PN model over SDF when possible in workflow design.

The workflow and experiment for a real ecology use case demonstrate its capability and usability.

Currently our Master-Slave architecture is implemented to utilize ad-hoc network computers. For future work, we plan to extend it to enable users to easily switch from local computation to other distributed computation environments, such as Cluster, Grid, and Cloud platforms.

An important part of our future work is to standardize third party data transfers among the distributed nodes. We're working with the rest of the Kepler community to consolidate different approaches.

We are in the process of conducting a study of distributed computing work within Kepler. We will categorize different distributed computing capabilities and, using use cases, we will explain how to choose a distributed computing approach to match the requirements of a problem.

8. Acknowledgements

The authors would like to thank the rest of the Kepler team for their collaboration. This work was supported by NSF SDCI Award OCI-0722079 for Kepler/CORE, NSF ITR Award No. 0225676 for SEEK and NSF CEO:P Award No. DBI 0619060 for REAP.

References

[1] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. Lee, J. Tao, Y. Zhao. "Scientific workflow management and the Kepler system". Concurrency and Computation: Practice and Experience, 18 (10), pp. 1039-1065. 2005.

[2] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver and K. Glover, M.R. Pocock, A. Wipat, and P. Li. "Taverna: a tool for the composition and enactment of bioinformatics workflows". Bioinformatics, 20(17), pp. 3045-3054, Oxford University Press, London, UK, 2004.

[3] I. Taylor, M. Shields, I. Wang, and A. Harrison. "The Triana Workflow Environment: Architecture and Applications". In I. Taylor, E. Deelman, D. Gannon, and M. Shields, editors, Workflows for e-Science, pp. 320-339. Springer, New York, Secaucus, NJ, USA, 2007.

[4] E. Deelman, G. Mehta, G. Singh, M. Su, and K. Vahi. "Pegasus: Mapping Large-Scale Workflows to Distributed Resources". In I. Taylor, E. Deelman, D. Gannon, and M. Shields, editors, Workflows for e-Science, pp 376-394. Springer, New York, Secaucus, NJ, USA, 2007.

[5] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. Seragiotto, Jr, and H. Truong. "ASKALON: a tool set for cluster and Grid computing". Concurrency and Computation: Practice and Experience, 17(2-4), pp. 143-169, Wiley InterScience, 2005.

[6] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, M. Wilde. "Swift: Fast, Reliable, Loosely Coupled Parallel Computation". Proceedings of 2007 IEEE Congress on Services (Services 2007), pp. 199-206, 2007.

[7] D. Abramson, C. Enticott and I. Altinas. "Nimrod/K: Towards Massively Parallel Dynamic Grid Workflows". Proceedings of Supercomputing 2008 (SC2008), Article No. 24.

[8] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, J. Good. "On the Use of Cloud Computing for Scientific Workflows". Proceedings of workshop SWBES08: Challenging Issues in Workflow Applications, 4th IEEE International Conference on e-Science (e-Science 2008), pp 640-645, 2008.

[9] J. Yu and R. Buyya. "A Taxonomy of Workflow Management Systems for Grid Computing". Journal of Grid Computing, 2006 (3), pp.171-200, 2006.

[10] H. Casanova and F. Berman. "Parameter Sweeps on The Grid With APST". In F. Berman, G. Fox and T. Hey, Grid Computing: Making the Global Infrastructure a Reality. Chapter 33, Wiley Publisher, Inc., 2002.

[11] E. Renshaw. "Modelling Biological Populations in Space and Time". vol. 11: Cambridge Studies in Mathematical Biology. Cambridge University Press, Chichester. 1991

[12] P. R. Hosseini. "Pattern Formation and Individual-Based Models: The Importance of Understanding Individual-Based Movement". Ecological Modeling 194: 357-371. doi:10.1016/j.ecolmodel.2005.10.041. 2006.

[13] P. R. Hosseini, A. Dobson and A. A. Dhondt. "Seasonality and wildlife disease: How seasonal birth, aggregation and variation in immunity affect the dynamics of Mycoplasma gallisepticum in House Finches". Proceedings of the Royal Society of London: Biological Sciences. 271:2569-2577. doi:10.1098/rspb.2004.2938. 2004.

[14] J. Wang, I. Altintas, C. Berkley, L. Gilbert, M. B. Jones. "A High-Level Distributed Execution Framework for Scientific Workflows". Proceedings of workshop SWBES08: Challenging Issues in Workflow Applications, 4th IEEE International Conference on e-Science (e-Science 2008), pp 634-639, 2008.

[15] I. Altintas, O. Barney, E. Jaeger-Frank. "Provenance Collection Support in the Kepler Scientific Workflow System". Proceedings of International Provenance and Annotation Workshop (IPAW2006), pp. 118-132, 2006.

[16] D. Thain, T. Tannenbaum and M. Livny, "Distributed Computing in Practice: The Condor Experience". Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, pp 323-356, February-April, 2005.

[17] I. Foster. "Globus Toolkit Version 4: Software for Service-Oriented Systems". IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2006.