Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 1 (2010) 1169–1178
Procedia Computer Science
www.elsevier.com/locate/procedia

International Conference on Computational Science, ICCS 2010

Theoretical enzyme design using the Kepler scientific workflows on the Grid

Jianwu Wang[a,*], Prakashan Korambath[b], Seonah Kim[c], Scott Johnson[c], Kejian Jin[b], Daniel Crawl[a], Ilkay Altintas[a], Shava Smallen[a], Bill Labate[b], Kendall N. Houk[c]

[a]San Diego Supercomputer Center, UCSD, 9500 Gilman Drive, MC 0505, La Jolla, CA 92093, U.S.A.
[b]Institute for Digital Research and Education, UCLA, 5308 Math Sciences, Los Angeles, CA 90095, U.S.A.
[c]Department of Chemistry and Biochemistry, UCLA, Los Angeles, CA 90095, U.S.A.

Abstract

One of the greatest challenges in computational chemistry is the design of enzymes to catalyze non-natural chemical reactions. We focus on harnessing the distributed parallel computational power of the Grid to automate the inside-out process of enzyme design using scientific workflow systems. This paper presents a scientific workflow based approach to facilitate the inside-out enzyme design process in the Grid execution environment by providing features such as resource consolidation, task parallelism, provenance tracking, fault tolerance and workflow reuse, which results in an automated, pipelined, efficient, extensible, stable, and easy-to-use computational process for enzyme design.

Keywords: enzyme design; computational chemistry; scientific workflow; Grid computing; pipeline; parallelism.

1. Introduction

Enzymes are nature's protein catalysts that accelerate and regulate all metabolic reactions. To design new catalysts computationally and then to make them with molecular biological techniques will be a breakthrough in technology. An *inside-out* design approach has been developed by Baker and Houk (see Section 2.1 for details). In the process, quantum mechanical calculations give a *theozyme* [1], or *theoretical enzyme*, which is theoretical optimum catalyst. Numerous protein scaffolds are then screened to determine which can be used to display the side chains to mimic the geometry of the theozyme; this procedure generates a large library of potential catalysts that are evaluated for fidelity to the theozyme. The best of these are subjected to mutations to obtain a design that will both fold and catalyze the reaction. The estimated execution task for one computation with a single scaffold would last about 2.7 days on a common desktop computer, generating approximately 1.1 GB output data. Typically, a theozyme with active sites is matched at least once per scaffold (226 protein scaffolds so far) to potentially accommodate the orientation of the model functional groups. The computation process needs to be repeated for

* Corresponding author. Tel.: +1-858-534-5110; fax: +1-858-534-8303.
E-mail address: jianwu@sdsc.edu.

1877-0509 © 2010 Published by Elsevier Ltd.
doi:10.1016/j.procs.2010.04.131

many times with different theozymes and calculation options. The goal of this work is to accelerate and facilitate this important computation- and data- intensive process for chemists.

A computational Grid is defined as "a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities" [2]. Embarrassingly parallel computations, like high-throughput job execution, are ideally suited for running on Grids. Yet the software that creates and manages Grid environments, such as Globus toolkit[b] and gLite[c], is not sufficient to manage the control and data dependency of jobs that are common for domain-specific problems. Building flexible, complex, and reusable computational chemistry calculation processes, like inside-out enzyme design process, requires combining more than one computational code [3][4]. Scientific workflow systems [5] would enable researchers to design computational experiments that span multiple computational and analytical models, and in the process, to store, access, transfer, and query information. This requires the integration of a variety of computational tools, including the chemistry software, preparation, visualization, and analysis toolkits, as well as database programs [3][4].

We already utilized Kepler scientific workflow system for computational chemistry on the Grid [3][6][7], which demonstrated its feasibility to combine and automate the high-throughput processing of computational chemistry tools from a unified interface. The implementation is based on the idea that the time-consuming parts of the computation can be distributed to computational Grids. In this paper, we present the recent progress on this effort, which improves our previous work from several perspectives, including task parallelism, provenance tracking, fault tolerance and portal environments. We will demonstrate how these improvements can make the inside-out enzyme design process more efficient, extensible, stable, and easy-to-use.

The paper is organized as follows. Section 2 describes the background of the inside-out enzyme design process, the Grid execution environment, and the Kepler scientific workflow system. Our implementation is described in Section 3 and execution experiments in Section 4. Finally, related work is discussed in Section 5 and conclusions and future work are presented in Section 6.

2. Background

2.1. Scientific Background

Enzymes are nature's catalysts that enhance the rates of reactions that are often too slow to sustain life. With accelerations of up to 10^{27}, enzymes achieve this catalytic proficiency by lowering the energy of the transition state, or the energy maximum on a reaction coordinate, of the rate-limiting step in a chemical reaction [8]. If a particular step in a chemical reaction has multiple transition states that lead to multiple products or intermediates, selectively stabilizing these transition states can lead to the rapid and selective formation of a desired product. Computing the structures and energies of transition states using quantum mechanical (QM) approaches has become a useful tool in studying the mechanisms and properties of enzyme-catalyzed reactions [9][10][11][12].

The inside-out design of enzymes to catalyze the chemical reactions for which there are no natural enzymes, also called non-natural chemical reactions, is a subject of intense interest. Two successes in this field involving the design and testing of novel Kemp eliminases and retro-aldolases have recently been reported [13][14]. These recent successes showed modest rate enhancements (4-6 orders of magnitude for the best designs) and rates of success (32/72 retro-aldolases and 12/59 Kemp eliminases showed any detectable levels of catalytic activity). The process used to design these enzymes is known as the inside-out approach, named because the entire enzyme designs are based on a truncated model of the transition-state stabilized by model amino acids, called a theozyme [1]. The entire inside-out design approach uses the following steps: 1) selection of target reaction and identification of its transition state; 2) design of a theozyme-model using QM calculations; 3) selection of a stable protein scaffold with an active site capable of accommodating the theozyme; 4) incorporation of the theozyme with geometric fidelity to produce an enzyme design; 5) computational screening and ranking of the catalytic activity of each enzyme design; and 6) experimental testing of the best designs. One big challenge of this process is the combinatorial explosion effect of steps 3-4 on the entire process. Numerous protein scaffolds are often able to accommodate a theozyme (sometimes

[b] http://www.globus.org/

[c] http://glite.web.cern.ch/glite/

the same scaffold in different spatial configurations), and from these many different final enzyme designs can be generated.

We have been developing computational methodology for enzyme design using quantum mechanics (QM) and molecular dynamics (MD). We locate structures of catalytic sites (theozyme) for the aromatic Claisen rearrangement of an allyl coumarin ether using density functional theory. The resultant theozymes are incorporated into existing stable protein scaffolds (226 scaffolds so far) using the Rosetta programs of Zanghellini *et al* [15]. The residues in the vicinity of active site are then optimized with RosettaDesign [16][17]. The designs are further evaluated using molecular dynamics.

2.2. University of California Grid

Computational Grids [2] are used to manage underlying distributed and heterogeneous computational resources and provide a uniform layer for parallel computation. The University of California (UC) Grid[d], also called UC Grid, is a Grid infrastructure comprised of participating computational clusters throughout the UC campuses. The UC Grid resources are open to all faculty, staff, and students who are members of any UC campus. The owners of the resources determine when and how much of those resources are allocated to the UC Grid resource pool. As shown in Fig. 1, the UC Grid architecture mainly consists of four parts:

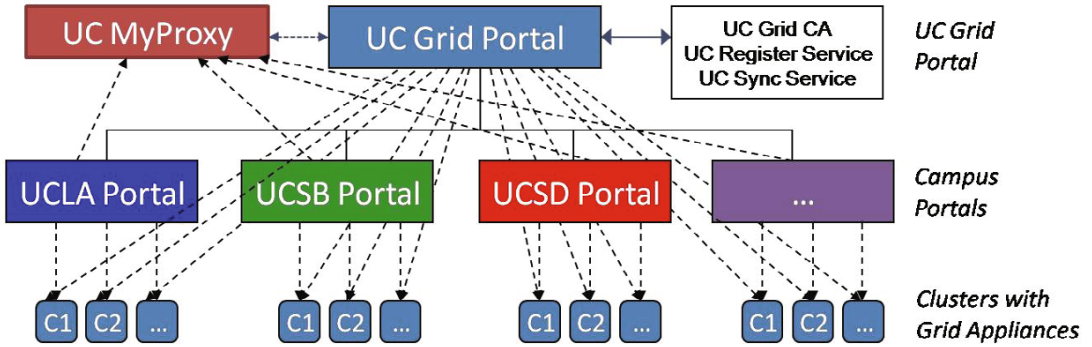

Fig. 1. The architecture of the University of California (UC) Grid.

The UC Grid Campus Portal. This is the web interface, front-end to the UC Grid for a single campus. It provides the user interface and serves as a single point of access for users to all campus computing resources. It communicates with the Grid Appliances in a single campus and also serves as an aggregation unit for Monitoring and Discovery Services (MDS) based resource discovery. Because the users are known only at the campus level, all applications for a Grid account starts at the campus portal. The campus portal takes the request from the users and authenticates them against Shibboleth based campus service and, if the authentication process is successful, sends a request to UC Grid portal to sign a X-509 certificate for the user through a web service called Register Service. When the process is done, the users can access to UC Grid Portal using their grid account issued by UC Grid portal.

The UC Grid Portal. This is the web interface where users can login to access all the clusters in all ten campuses. This is the super portal of all the UC Grid campus portals and is the certificate signing authority for the entire UC Grid. Once the certificate is signed, it is pushed to UC MyProxy server that leases the short-lived credential every time a user login to the Grid portal. Any changes of account and resource status on one campus portal will be updated on the UC Grid portal immediately through a web service called Sync Service.

The UC MyProxy Server. The primary purpose of this server is to store user credentials and release them to UC Grid portal when users login to the portal.

The Grid Appliances. The Grid Appliances at one cluster enable that cluster to participate in the UC Grid. Grid appliance in practice is a parallel head node of the cluster, with Grid software and credentials installed. The addition of a Grid appliance in no way modifies policy decisions on the cluster level. Every account in UC Grid has a distinguished name (DN Name). This DN Name has to be mapped to a local account on a cluster so that grid

[d] http://www.ucgrid.org/

requests such as job submission and file transfer are done as a valid user on that cluster. So, during the account registration process, UC Grid informs the cluster administrator with DN Name of the user and the cluster administrator in turn has to enter a new entry to the grid-mapfile in the Grid appliance node before approving it. The UC Grid will then let users know that they can start using the new resource (cluster) from the Grid portal.

2.3. Kepler Scientific Workflow System

Scientific workflow management systems [5] 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.

The Kepler project[e] aims to produce an open source scientific workflow system that allows scientists to easily design and efficiently execute scientific workflows. Inherited from Ptolemy II[f], Kepler adopts the actor-oriented modeling [18] paradigm for scientific workflow design and execution. Each actor is designed to perform a specific independent task that can be implemented as *atomic* or *composite*. Composite actors, or sub-workflows, are composed of atomic actors bundled together to perform complex operations. Actors in a workflow can contain *ports* to consume or produce data, called *tokens*, and communicate with other actors in the workflow through communication channels via *links*.

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 *director*. The director defines how actors are executed and how they communicate with each other. Since the director is decoupled from the workflow structure, a user can easily change the computational model by replacing the director using the Kepler graphical user interface. As a consequence, a workflow can execute sequentially, e.g., using the Synchronous Data Flow (SDF) director, or in parallel, e.g., using the Process Network (PN) director.

Kepler provides an intuitive graphical user interface and an execution engine to help scientists to edit and manage scientific workflows and their execution. In the Kepler GUI, actors are dragged and dropped onto the canvas, where they can be customized, linked, and executed. Further, the execution engine can be separated from the user interface thereby enable the batch mode execution.

Currently, there are over 200 actors available in Kepler, which largely simplify the workflow composition. We will briefly describe the main distinctive actors that are used in our approach.

Local Execution Actor. The *ExternalExecution* actor in Kepler is the wrapper for executing commands that run legacy codes in a workflow. Its purpose is to call the diverse employed external programs or corresponding automatically generated temporary shell-script wrappers.

Job Submission Actors. Job submission is a common and efficient way to achieve parallelism on distributed computing resources. Kepler provides two sets of actors that can submit jobs to two typical distributed resources: Cluster and Grid. Each set has actors to be used for different job operations, e.g. create, submit, and status check.

Data Transfer Actors. Access and management of remote data are basic functions in distributed Grid computing. There are multiple sets of data transfer actors in Kepler to support moving data from one location to another by different ways, e.g., FTP, GridFTP, scp, and others.

Fault Tolerance Actor. Fault tolerance provides robust execution, which is especially useful in loosely-coupled distributed computation environments. The fault tolerance actor, which is a composite actor that contains multiple sub-workflows, supports automatic exception catching and handling by re-trying the default sub-workflow or executing the alternative sub-workflows.

Besides the above actors, Kepler also provides the following characteristic capabilities for workflow execution, which will benefit the enzyme design process:

Inherent and Implicit Parallelism. Kepler supports inherent and implicit parallelism since it adopts a dataflow modeling approach [19]. Actors in Kepler are independent from each other, and will be triggered once their input data are available. Since explicit parallel primitives, such as parallel-for, are not needed, workflow composition is

---

[e] http://www.kepler-project.org/

[f] http://ptolemy.eecs.berkeley.edu/ptolemyII/

greatly simplified. The workflow execution engine will parallelize actor execution automatically at runtime according to their input data availability.

Pipeline Parallelism. For a sequence of multiple actors that need to be executed sequentially, we can still realize a certain parallelism, called *pipeline parallelism* [20], if the input data is a set of tokens. The execution of the tokens in the token set can be independent and parallel. Kepler supports pipeline parallelism by token streaming, blocking and buffering techniques [20].

Provenance Tracking. Users may need to track workflow output data generated by domain specific programs according to given input parameter values. Kepler provenance framework [21] supports collection and query on workflow structure and execution information in both local and distributed environments [22].

3. Our Approach for Automatic Theoretical Enzyme Design Process

3.1. Overall Cyberinfrastructure Architecture for Theoretical Enzyme Design

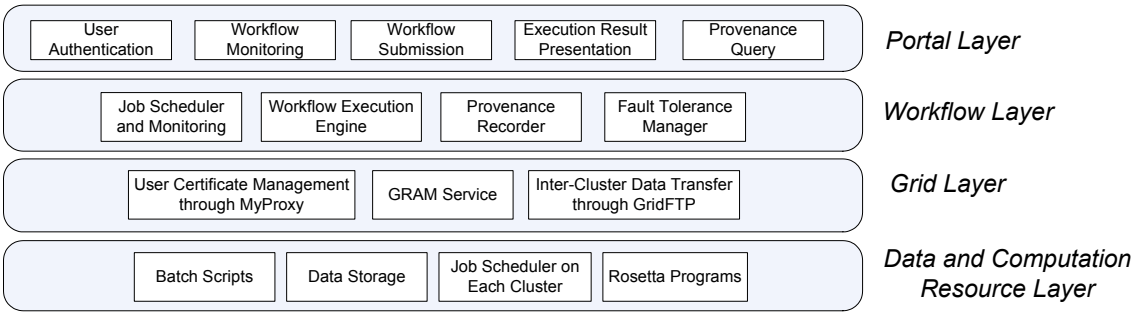

Fig. 2. Overall cyberinfrastructure architecture for the theoretical enzyme design process.

The overall cyberinfrastructure architecture for the theoretical enzyme design process is presented in Fig. 2, which consists of four layers: 1) the Portal layer for web-based user interaction; 2) the Workflow layer for enzyme design process automation; 3) the Grid layer to manage multiple clusters and their interaction; and 4) the Data and Computation Resource layer where the enzyme design programs and data located on clusters. In the following sub-sections, we will mainly explain how the workflow layer can help the enzyme design process automation.

3.2. Conceptual Enzyme Design Workflow

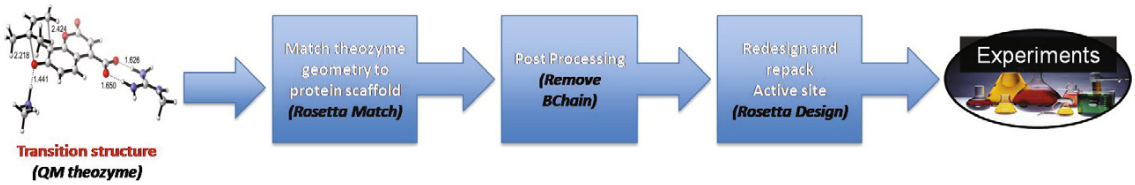

Fig. 3. Conceptual workflow of enzyme protein design process.

As shown in Fig. 3, the conceptual enzyme design workflow takes QM theozymes as inputs, and goes through three discrete steps before validation by experiments. The goal of this conceptual workflow is to standardize the enzyme design process through automation, and eliminate the need for unnecessary human interaction. Sequences of tasks in the enzyme design process are repeated using the same series of programs, which must be executed to design and evaluate an enzyme for a new chemical reaction. For example, a theozyme with active sites is matched at least once per scaffold (total 226 scaffolds so far) to potentially accommodate the orientation of the model functional groups.

The entire enzyme design process can be performed independently for different scaffolds, and the total computation time for each scaffold can vary. The number of matches per theozyme is about 100-4,000 and computation time for a single scaffold on one CPU core is usually 1-3 hours. The number of enzyme designs generated by RosettaDesign per match is about 100-15,000 and computational time on one CPU core is usually 0.5-

2 hours. One whole computation time required for all 226 scaffolds could take months on one single computer core and the total number of generated enzyme designs is about 7 million.

3.3. Enzyme Design Workflow in Kepler for Inner-Cluster Parallel Execution

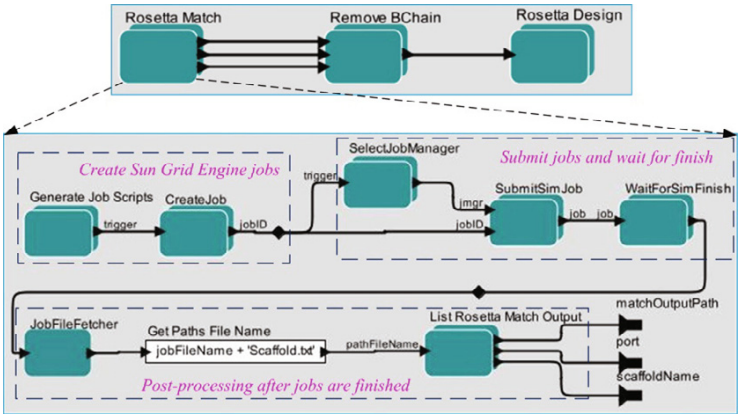

Fig. 4. Kepler workflow for Enzyme Design Processes on One Cluster.

As shown in Fig. 4, a workflow is made to utilize the tens to hundreds of computing CPU cores in one cluster. The top-level workflow structure is the same as the conceptual workflow in the previous sub-section. Inside a composite actor, such as RosettaMatch, the sub-workflow dynamically creates job scripts according to a user' inputs and submits them to the Sun Grid Engine job scheduler on the cluster using Kepler job actors. By sending jobs for all scaffolds to the job scheduler on the cluster, the jobs can be concurrently executed on the nodes of the cluster and the concurrency capability is determined by the node capacity of the cluster.

Many jobs will be executed through one composite actor, for example one job will be submitted in the RosettaMatch actor and 1-30 independent jobs in the RosettaDesign actor for each scaffold. Once a job is finished, it will trigger downstream actors, such as RemoveBChain actor, which post-process the output data from RosettaMatch. Kepler can naturally express the dependencies using connection links among actors and no explicit primitive or structure is needed to express the parallel. Using the PN director, all actors will be executed independently based on their input data availability, which can realize automatic parallel computation of these actors.

With pipeline parallel support in Kepler, the workflow input can be a vector of data, which greatly simplifies workflow construction. Each workflow execution will process many scaffolds, so we list the scaffold directory to create a vector of scaffolds and read it as input to the workflow rather than having many explicit parallel branches or loops on whole workflow. Each scaffold does not need to wait for the finish of the RosettaMatch step on other scaffolds. It also fits that different scaffolds finish their execution within very different times. Additionally the sequence capacity can dynamically change during the workflow execution. For example, RosettaMatch will can generate anywhere between 100-4,000 matches, for different scaffolds, and these matches will dynamically be used as the input elements for the next steps in the workflow.

3.4. Enzyme Design Workflow in Kepler for Inter-Cluster Parallel Execution

By adopting Globus as the Grid security and service infrastructure, the workflow shown in Fig. 5 easily can be executed in parallel among multiple clusters. For a local cluster, we execute the Kepler workflow shown in Fig. 4 through the Globus GRAM service deployed on the cluster. For remote clusters, two extra tasks need to be done besides the workflow execution: 1) Input data needs to be staged in to the remote clusters before workflow execution and 2) output data needs to be staged out from remote cluster back to local cluster. We employ GridFTP to do data stage in and out. The computations on multiple clusters are independent of one another, so there are no control or data flow connections among the actors for these executions in the workflow, and the Kepler engine will

run them in parallel. The two-level workflow makes previous workflows for one cluster easy to be reused. One challenge for this inter-cluster workflow is how to optimize the load balance on multiple clusters and the data stage in/out overhead.

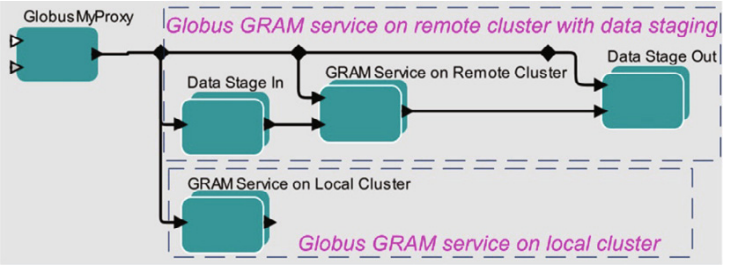

Fig. 5. Kepler workflow for enzyme design processes on multiple clusters. The GRAM service composite actor will invoke the workflow shown in Fig. 4 on the target cluster.

3.5. Provenance and Fault Tolerance Support in the Workflow

While using more clusters increases our computational capacity, it also increases our probability of failure. Typical failures include: 1) nodes selected for a job fail while the job is still running; 2) the cluster job scheduler fails to start the job; 3) a whole cluster may be down; 4) a user's credential may expire during the execution time; 5) the Globus GRAM service fails. Although these exceptions happen occasionally, the whole execution of the enzyme design workflow will crash without fault tolerance support.

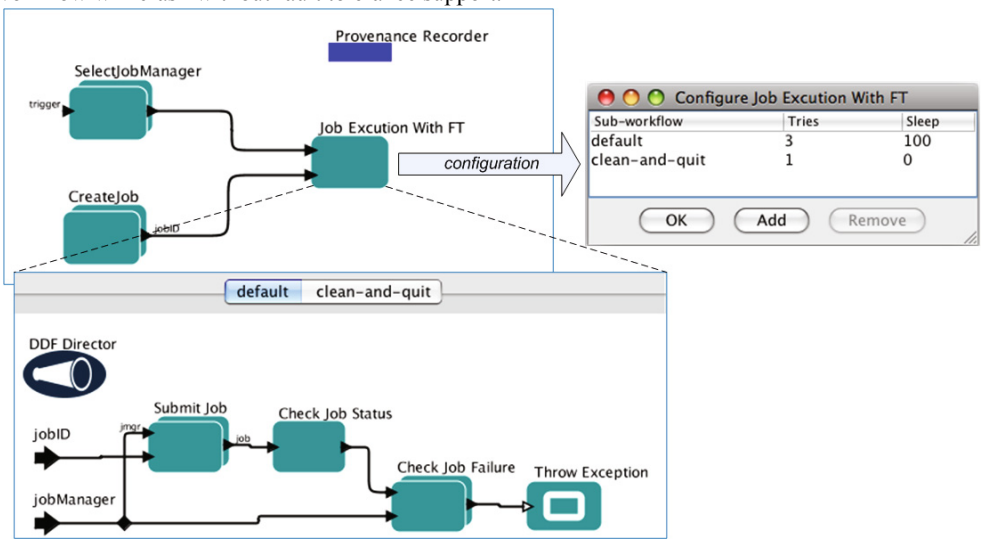

Fig. 6. Fault tolerance and provenance support in Kepler workflow.

To support fault tolerance at workflow level, we adopt the Fault Tolerance (FT) actor for some sub-workflows. A simplified example is shown in Fig. 6, where the *JobExecutionWithFT* actor is a customized fault tolerance actor. There are two sub-workflows (shown in the bottom part of Fig. 6) in the actor, namely *default* and *clean-and-quit*, which will be invoked according to the configurations of the JobExecutionWithFT actor. The default sub-workflow submits a job, checks the job status, and throws an exception if the job fails. After catching the exception, the JobExecutionWithFT actor will re-execute the default sub-workflow after sleeping for 100 seconds. If the default sub-workflow still gets exceptions after three retries, the clean-and-quit sub-workflow, which cleans intermediate data and stops the whole execution, will be executed with the same input tokens.

The exceptional handling logic can be easily expressed with the fault tolerance actor; no explicit loop and conditional switch is needed in the workflow. Further, the input data for sub-workflow retry do not need to be

explicitly saved since they are automatically recorded by Provenance Recorder and will be fetched for re-execution.

Besides fault tolerance, Kepler provenance also supports collection and query on workflow structure and execution information in both local and distributed environments [22]. Each enzyme design workflow execution will generate millions of designs and chemists may need the workflow to be executed for many times with different input models. Kepler provenance can help chemists to track the data efficiently in the future, such as querying which input model was used to generate one particular design.

4. Workflow Execution and Experiments

4.1. Workflow Execution from UC Grid Portal

To hide the unnecessary implementation details and make it easier to use for chemists, Kepler is being integrated into UC Grid portal as an application resource. A user first selects the Kepler workflow to be executed on the UC Grid. Then the input parameters for the chosen workflow will be displayed for the user to fill in. After the user configures the parameters, the workflow is ready to be deployed on the UC Grid.

When the user submit the workflow, UC Grid portal will first search the information of available clusters in the Grid and know which clusters have Kepler and the application in the workflow (such as RosettaMatch) available. Then UC Grid portal will automatically transfer the files from the user specified location to the best available cluster, and submit a Globus GRAM job to run the workflow on the cluster. If the workflow needs to be run on multiple clusters, the parameters for remote clusters in the workflow need to be dynamically specified by UC Grid portal. The GRAM job will be run on the user's account if the user has an account on the chosen cluster. Otherwise the job will be run on a pre-existing guest user account in the clusters created exclusively for UC Grid jobs. The Grid will monitor the completion of its workflow execution using scheduler event generator. Users can download the output files when the status becomes 'Done'. Users can submit a workflow from either UC Grid portal or UC Grid campus portal, the difference is that the submission from UC Grid portal can utilize all available clusters on all campuses whereas the submission from campus portal can only utilize available clusters on that campus.

4.2. Experiments

To measure the speedup capabilities of the workflows, we experimented on two clusters where Globus 4.0.7 and Sun Grid Engine job scheduler are deployed. Cluster 1 has 8 nodes, each with 24 GB of memory and two 2.6 GHz quad-core CPUs. Cluster 2 has 30 nodes, each with 4 GB of memory and two 2.2 GHz single-core CPUs.

Our first experiment executed the enzyme design workflow described in Section 3.3 on cluster 2 with different usable CPU core configurations. We also executed another workflow that only had the RosettaMatch part of the enzyme design workflow in order to determine the speedup difference. We ran the workflows with different inputs. As shown in Table 1, all these tests, no matter the differences of workflow structures and inputs, have good scalability and speedup when the usable core number grows.

Table 1. Workflow execution with different usable CPU cores.

workflow	workflow Input	output data	total job number	workflow execution time (unit: hour)		
				1 core	25 core	60 core
RosettaMatch	10 scaffold	0.29 GB	10	3.38	0.69	0.63
RosettaMatch	226 scaffold	27.96 GB	226	128.61	5.52	3.06
RosettaMatch + RemoveBChain + RosettaDesign	10 scaffold	10.92 GB	296	533.61	29.32	11.24

We also tested the enzyme design workflow in Section 3.3 and 3.4 to know the concurrence performance on two clusters. From the execution data of the workflow execution only on the cluster 1 and 2, we can know cluster 1 is about twice as fast. So approximately twice as many inputs are distributed to cluster 1 and cluster 1 is set as local cluster when the workflow is executed on the two clusters. The experiment data, shown in Table 2, demonstrates the good concurrence performance in the second and third test. The poor performance in the first test is because there are too few jobs comparing to the core number. We can also see the speedup ratios are not as good as those in the first experiment. The reasons are twofold: 1) it is hard to realize good load balance on multiple clusters since the execution time for different scaffold varies; 2) the data stage in and out phases for remote cluster may cause a big

overhead if the transferred data is very large.

Table 2. Workflow execution with different clusters.

workflow	workflow input	total job number	workflow execution time (unit: hour)		
			cluster 1 (64 core)	cluster 2 (60 core)	cluster 1 and 2
RosettaMatch	10 scaffold	10	0.33	0.63	0.36
RosettaMatch	226 scaffold	226	1.52	3.06	1.33
RosettaMatch + RemoveBChain + RosettaDesign	10 scaffold	296	6.17	11.24	4.21

5. Related Work

There have been many efforts to utilize Grid computing to accelerate chemistry applications [23][24][25]. The Grid portal in [23] facilitates the execution of computational quantum chemistry and molecular dynamics software in Grid environments. The OpenMolGRID system mainly includes three main application components: data warehousing, data mining, and molecular engineering [24]. The Computational Chemistry Grid system leverages Grid middleware to provide an easy-to-use integrated computing environment of chemistry software packages on supercomputer resources [25]. Neither system contains a workflow layer, making it challenging to automate chemistry processes and optimize using parallelism in these environments.

There have also been several projects utilizing scientific workflow systems to realize process automation in Grid environments [5]. The Pegasus system dynamically maps abstract workflows onto Grid resources [26]. The ASKALON toolkit focuses on Grid workflow application optimization and fault tolerance [27]. Swift is targeted to reliable and efficient execution of large loosely coupled computations on Grid environments [28]. Yet as far as we know, none of these support all aspects of portal support, implicit parallel, pipeline parallel, fault tolerance and provenance tracking, which are important in computational chemistry process automation in Grid environments.

6. Conclusion and Future Work

Domain scientists greatly benefit from the enhanced capability and usability of cyberinfrastructure. This paper explains how the Kepler scientific workflow system can facilitate inside-out enzyme design process in Grid environments by providing features including resource consolidation, parallelism, provenance tracking, fault tolerance and workflow reuse. Our implementation and experiments demonstrate these features can make the enzyme design computation automated, pipelined, efficient, extensible, stable, and easy-to-use.

We are working on an EDGE (Enzyme Design Geometry Evaluation) program, which compares the geometrical similarity between the catalytic residues/substrates in the original theozyme and the candidate enzyme designs created from Rosetta program. EDGE will greatly filter out useless results of RosettaMatch and RosettaDesign by geometrical filtering based on the theozyme structure. Additionally, we plan to improve load balance capability of workflow execution on multiple clusters based on cluster capacities and workflow execution history data.

Acknowledgements

The authors would like to thank the rest of the Kepler and UC Grid community for their collaboration. This work was supported by NSF SDCI Award OCI-0722079 for Kepler/CORE, NSF CEO:P Award No. DBI 0619060 for REAP, DOE SciDac Award No. DE-FC02-07ER25811 for SDM Center, and UCGRID Project. We also thank the support to the Houk group from NIH-NIGMS and DARPA.

References

1. D.J. Tantillo, J. Chen, and K.N. Houk, Theozymes and compuzymes: theoretical models for biological catalysis. Curr Opin Chem Biol, 1998. 2(6): pp. 743-50.

2. I. Foster and C. Kesselman, Computational Grids. In I. Foster and C. Kesselman, eds., The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufman (1999), pp. 15-51.

3. W. Sudholt, I. Altintas and K. Baldridge, Scientific Workflow Infrastructure for Computational Chemistry on the Grid. In Proceedings of

the 1st Computational Chemistry and Its Applications Workshop at the 6th International Conference on Computational Science (ICCS 2006). LNCS 3993, pp. 69-76. 2006.

4. A. Tiwari and A.K. T. Sekhar, Workflow based framework for life science informatics. Computational Biology and Chemistry 31(5-6), pp. 305-319. 2007.

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

6. K.K. Baldridge, J.P. Greenberg, W. Sudholt, S. Mock, I. Altintas, C. Amoreira, Y. Potier, A. Birnbaum, K. Bhatia, M. Taufer, The Computational Chemistry Prototyping Environment, Proceedings of the IEEE, 93 (2005), pp. 510-521.

7. K.K. Baldridge, W. Sudholt, J.P. Greenberg, C. Amoreira, Y. Potier, I. Altintas, A. Birnbaum, D. Abramson, C. Enticott, S. Garic, Cluster and Grid Infrastructure for Computational Chemistry and Biochemistry. In A.Y. Zomaya, ed., Parallel Computing for Bioinformatics and Computational Biology, Wiley (2005), pp. 533-552.

8. X. Zhang and K.N. Houk, Why enzymes are proficient catalysts: beyond the Pauling paradigm. Acc Chem Res, 2005. 38(5): pp. 379-385.

9. S.J. Benkovic and S. Hammes-Schiffer, A perspective on enzyme catalysis. Science, 2003. 301(5637): pp. 1196-1202.

10. M. Garcia-Viloca, J. Gao, M. Karplus, D.G. Truhlar, How enzymes work: analysis by modern rate theory and computer simulations. Science, 2004. 303(5655): pp. 186-195

11. C.L. Stanton, I.W. Kuo, C.J. Mundy, T. Laino, K.N. Houk, QM/MM metadynamics study of the direct decarboxylation mechanism for orotidine-5'-monophosphate decarboxylase using two different QM regions: acceleration too small to explain rate of enzyme catalysis. J Phys Chem B, 2007. 111(43): pp. 12573-12581.

12. A. Warshel, Computer simulations of enzyme catalysis: methods, progress, and insights. Annu Rev Biophys Biomol Struct, 2003. 32: pp. 425-243.

13. L. Jiang, E.A. Althoff, F. R. Clemente, L. Doyle, D. Rothlisberger, A. Zanghellini, J.L. Gallaher, J.L. Betker, F. Tanaka, C.F. Barbas III, D. Hilvert, K.N. Houk, B.L. Stoddard, D. Baker, De novo computational design of retro-aldol enzymes. Science, 2008. 319(5868): pp. 1387-1391.

14. D. Rothlisberger, O. Khersonsky, A.M. Wollacott, L. Jiang, J. DeChancie, J. Betker, J.L. Gallaher, E. A. Althoff, A. Zanghellini, O. Dym, S. Albeck, K.N. Houk, D.S. Tawfik, D. Baker, Kemp elimination catalysts by computational enzyme design. Nature, 2008. 453(7192): p. 190-195.

15. A. Zanghellini, L. Jiang, A.M. Wollacott, G. Cheng, J. Meiler, E.A. Althoff, D. Röthlisberger, D. Baker, New algorithms and an in silico benchmark for computational enzyme design. Protein Sci. 15(12), 2006. pp. 2785-2794

16. G. Dantas, B. Kuhlman, D. Callender, M. Wong, D. Baker, A Large scale test of computational protein desing: Folding and stability of nine completely redesigned globular proteins. J. Mol. Biol. 332(2), pp. 449-460 (2003).

17. J. Meiler, D. Baker, ROSETTALIGAND: Protein-small molecule docking with full side-chain flexibility. Proteins 65, pp 538-548 (2006).

18. 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.

19. E.A. Lee and T. Parks, Dataflow Process Networks. Proceedings of the IEEE, 83(5), pp. 773–799, May 1995.

20. C. Pautasso, G. Alonso, Parallel Computing Patterns for Grid Workflows, In: Proc. of the HPDC2006 Workshop on Workflows in Support of Large-Scale Science WORKS06, Paris, France, June 2006.

21. 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.

22. J. Wang, I. Altintas, P.R. Hosseini, D. Barseghian, D. Crawl, C. Berkley, M. B. Jones. Accelerating Parameter Sweep Workflows by Utilizing Ad-hoc Network Computing Resources: an Ecological Example. In Proceedings of IEEE 2009 Third International Workshop on Scientific Workflows (SWF 2009), 2009 Congress on Services (Services 2009), pp. 267-274, 2009.

23. Z. Zhou, F. Wang, B. D. Todd: Development of Chemistry Portal for Grid-enabled Molecular Science. In Proceedings of the First International Conference on e-Science and Grid Computing (e-Science'05), pp. 48-55, 2005.

24. S. Sild, U. Maran, A. Lomaka, M. Karelson, Open Computing Grid for Molecular Science and Engineering, J. Chem. Inf. Model., 46, pp. 953-959, 2006.

25. R. Dooley, K. Milfeld, C. Guiang, S. Pamidighantam, G. Allen: From Proposal to Production: Lessons Learned Developing the Computational Chemistry Grid Cyberinfrastructure. J. Grid Computing, 4(2), pp.195-208 (2006).

26. 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.

27. 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.

28. 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.