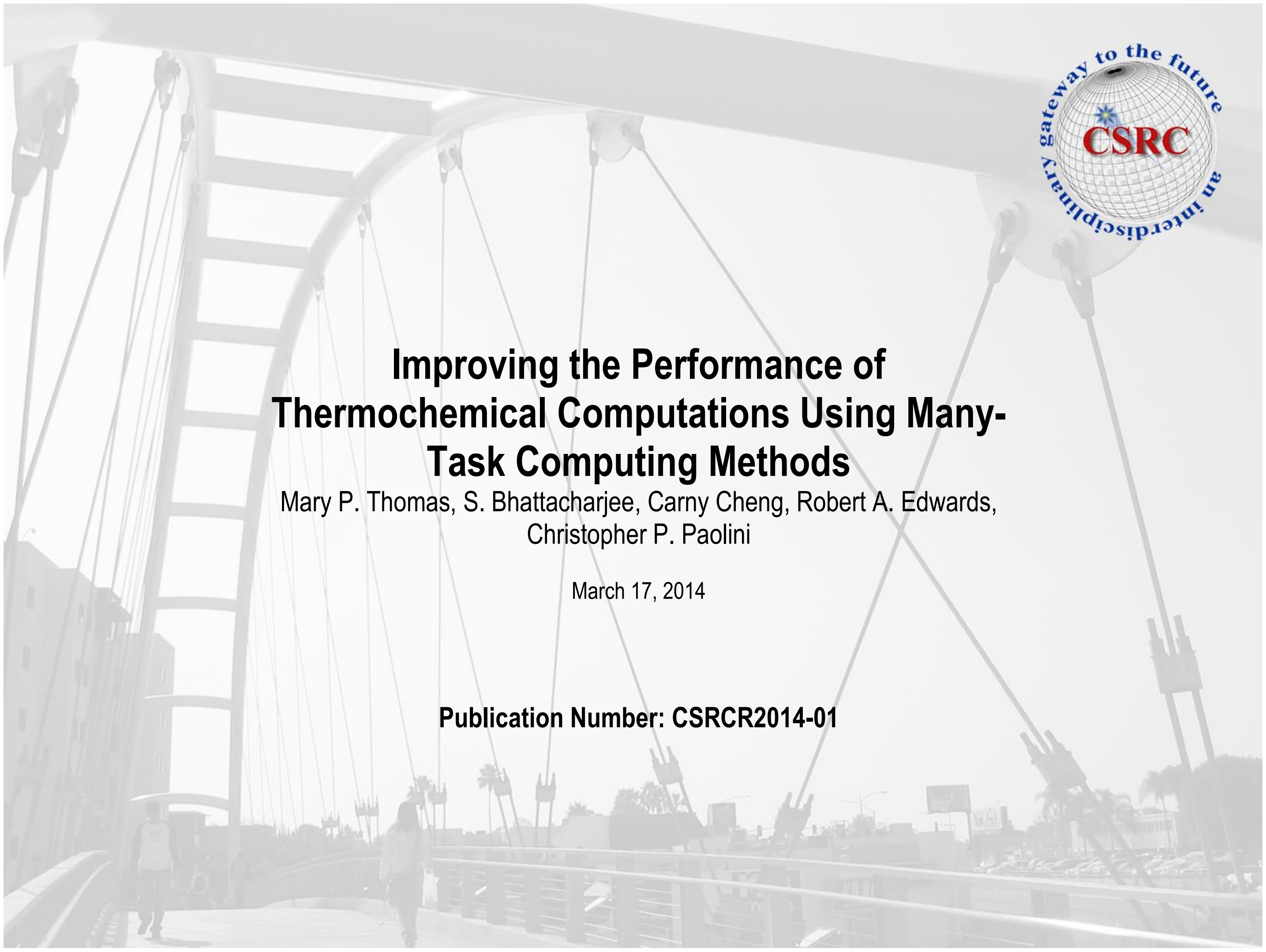


Improving the Performance of Thermochemical Computations Using Many-Task Computing Methods

Mary P. Thomas, S. Bhattacharjee, Carny Cheng, Robert A. Edwards, Christopher P. Paolini

March 17, 2014

Publication Number: CSRCR2014-01

Improving the Performance of Thermochemical Computations Using Many-Task Computing Methods

Mary P. Thomas, Member IEEE, S. Bhattacharjee Carny Cheng, Robert A. Edwards, Christopher P. Paolini

Abstract—The SDSU online Chemical Equilibrium Services perform numerical heat transfer and fluid flow computations, using the Flame3D simulator, for thousands of researchers, educators, and students. The computation is broken down into a grid of 2D or 3D control volumes, each of which runs for a few seconds, has small memory requirements (100 Bytes), is independent of its neighbors, and is submitted individually to a Web Service. The embarrassingly parallel simulation requires several hours to compute a few thousand control volumes, for 10's of thousands of iterations on a desktop. To improve the computational performance, a multi-task computing (MTC) approach was adopted. For this, a simple job distribution Web service framework (JODIS) was designed that distributes application workloads across hetergenous computing systems. JODIS has been demonstrated to run millions of Flame3D tasks simultaneously on a variety of resources and queuing systems. In this paper we report on the impact of JODIS on Flame3D computations, along with our experiences gained and challenges encountered when using heterogeneous computing environments, including the TeraGrid. Using JODIS, we have demonstrated a significant increase in the resolution of Flame3D (from 10^3 to more than 10^6 control volumes) and significant reduction in run times (by a factor of over 40 for a large test case of 128 processors and 10^7 tasks). In general, we conclude that the MTC approach can significantly improve Flame3D computational performance, but that changes need to be made to queuing/job submission systems in order to facilitate the rapid cycles needed for jobs similar to the Flame3D tasks.

Index Terms—Many-task computing, MTC, high-throughput computing, HTC, high performance computing, HPC, Web services, Pylons.

1 Introduction

The Chemical Equilibrium Services (CHEQS) application, created at San Diego State University (SDSU), solves chemical equilibrium problems for thousands of researchers, educators, and students [1]. The application does this using a novel numerical method to minimize a system's Gibbs free energy function with constraints [2]. CyberCHEQS is based on the TEST project, which has been in existence for several years, and has a large user community of several thousand users distributed across many educational institutions. TEST provides a Web interface that is used to analyze thermofluids problems, verify hand calculations, pursue what-if scenarios, and visualize thermal systems. TEST utilizes Web Services to provide easy to use software tools that allow engineers to upload data and run experiments and is able to facilitate online collaboration among users from diverse scientific backgrounds. The cyberCHEQS application (which is described in Section 2) is designed to compute 10^3 to 10^6 control volumes running for about 500,000 iterations. Currently, the system runs chemical computations on 2D or 3D volumes, consisting of 10^3 to 10^6 control volumes (i.e. cells of a grid) and submits 1 calculation at a time to a Web Service. Depending on the granularity of the computation, the system is currently limited to running small tasks (e.g. a few thousand control volumes for 10's of thousands of iterations). New approaches are needed in order to reduce time to solutions, increase resolution, and improve real-time simulations.

To improve the computational performance, we adopted a multi-task computing (MTC) approach that must operate within the heterogeneous computing environments typically encountered at a research university. For this, we designed a simple job distribution Web service framework (JODIS) that distributes application workloads across heterogeneous computing systems. The design and archticture are presented in Section 3 below. JODIS is a Web Service application framework based on the master/worker design pattern and utilizes the SDSU Cyberinfrastructure Web Application Toolkit (cyberWeb) as a hosting environment [3] [4]. JODIS has been demonstrated to run millions of Flame3D tasks simultaneously on a variety of resources and queuing systems. Using JODIS, we have demonstrated a significant increase in the

- *Mary Thomas is with the Department of Computer Science and the Computational Sciences Research Center, San Diego State University, San Diego, California, 92182. E-mail: mthomas@sciences.sdsu.edu.*
- *Cheng is with the Computational Sciences Research Cente,San Diego State University, San Diego, California, 92182. E-mail: carny@me.com.*
- *Edwards is with the Department of Computer Science and the Computational Sciences Research Center, San Diego State University, San Diego, California, 92182. E-mail: redwards@cs.sdsu.edu.*
- *Paolini is with the Departmenst of Computer Science and the Computational Sciences Research Center, San Diego State University, San Diego, California, USA. Email: paolini@engineering.sdsu.edu.*

Manuscript received Junary 4, 2010.

resolution of Flame3D (from 10^3 to more than 10^6 control volumes) and significant reduction in run times (by a factor of over 40 for a large test case of 128 processors and 10^7 tasks). In general, we find that the MTC approach can significantly improve the CyberCHEQS/Flame3D computational performance, but that changes need to be made to queuing/job submission systems in order to facilitate the rapid cycles needed for jobs similar to the Flame3D tasks tasks. These and other problems and issues encountered which limited our ability to run MTC problems on clusters and standard HPC clusters are discussed in more detail in the Conclusions Section of this paper.

2 The SDSU Chemical Equilibrium Services

The SDSU Chemical Equilibrium Services (CHEQS) have been designed as a Web based alternative to commonly used software applications such as STANJAN and NASA CEA for performing chemical equilibrium analysis in combustion research [5]. The approach is based on the next generation style of computational software development that relies on loosely-coupled network accessible software components called Web Services. While several projects in existence use Web Services to wrap existing commercial and open-source tools to mine thermodynamic data, no Web Service infrastructure has yet been developed to provide the thermal science community with a collection of publicly accessible remote functions for performing complex combustion computation. This work represents the first effort in the thermochemisty community to provide an infrastructure where remotely accessible software services allows developers of thermodynamics and combustion software to perform complex, multiphase chemical equilibrium computation with relative ease. This service can be integrated into any numerical application an example where we have coupled the use of our equilibrium service with an existing thermal-fluid simulator based on the control-volume formulation of Patankar and Spalding. In this section we provide an overview of chemical equilibrium computation, we describe the CHEQS Web services, as well as a description of the Flame3D CHEQS-based application, which is used for the research reported in this paper.

2.1 Background: Chemical Equilibrium Computation

Numerical determination of the equilibrium state mass fractions of gaseous and condensed matter is frequently needed in combustion simulations that model chemically reacting flows. The numerical method most often used to calculate an equilibrium distribution is based on minimizing a system's Gibbs free energy function with constraints. The total Gibbs energy G of a system composed of *m* species is given by

$$G = \sum_{j=1}^{m} \mu_j n_j \tag{1}$$

where $\mu_j = \mu_j\left(T, P, \mu_{i \neq j}\right)$ is the chemical potential of the jth species and a function of the system temperature T, pressure P, and number of moles of each component species, $n_{n \neq j}, \forall i$. From (1) and using the definition of the chemical potential for an ideal gas species j, we have

$$\mu_j = \mu_j^o + RT \ln\left(P_j / P^o\right) \tag{2}$$

where P^o is the standard state pressure of 1 atm and P_j the partial pressure of species j. The minimum stationary point of (1) will be the vector of species molar values $\vec{n}$ where dG vanishes. Differentiating (1) we obtain

$$dG = \sum_{j=1}^{m} n_j d\mu_j + \sum_{j=1}^{m} \mu_j dn_j \tag{3}$$

From the isothermal, isobaric Gibbs-Duhem equation we know that

$$\sum_{j=1}^{m} n_j d\mu_j = 0 \tag{4}$$

and so we seek the unique vector $\vec{n}^*$ such that

$$\sum_{j=1}^{m} \left[\left(\mu_j^o / RT\right) + \ln n_j - \ln n + \ln\left(P/P^o\right)\right] dn_j = 0 \tag{5}$$

where n_j is the number of moles of species j and $n = \sum_{j=1}^{m} n_j$ is the total number of moles in the equilibrium composition. Solving (5) amounts to solving a nonlinear constrained minimization problem. The traditional numerical practice used for solving optimization problems of this type is the method of Lagrange multipliers using an iterative Newton-Raphson technique for solving the resulting set of nonlinear equations. Bhattacharjee and Paolini [2] have encapsulated this method into a publicly accessible Web Service that can be called from most contemporary programming environments and commercial applications. This service can be invoked as a reusable third party software component by a thermal science researcher when developing custom applications. As a result, the researcher is freed from having to worry about implementing his or her own code for computing chemical equilibrium. When a desired equilibrium distribution is needed, the developer need only insert the requisite code to remotely discover and dynamically invoke the Web Service.

2.2 Equilibrium Computation Using the CHEQS Web Service

Web Services extend the paradigm of object-oriented programming to the network whereby a single software application is composed of loosely-coupled modules that execute on autonomous networked systems. Recent efforts by Dong et al. [6] and Truong et al. [7] have shown the strength of using Web Service technology in chemoinformatics applications to facilitate the organization and retrieval of chemical data while advances in collaborative cyberinfrastructure for developing predictive models for chemically reacting systems have been spearheaded by Frenklach et al. through the open-source Process Informatics Model (PrIMe) project [8]. Furthermore, the Cantera [9] package by Goodwin provides an open-source, object-oriented suite of software tools to aid in simulating

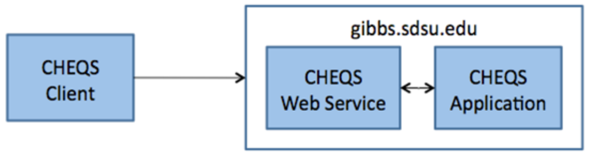


Figure 1. CHEQS client-server architecture. [3]

be called to solve equilibrium problems from within FORTRAN, MATLAB, and Python scripts. While several projects in existence use Web Services to wrap existing commercial and open-source tools to mine thermodynamic data, no Web Service infrastructure has yet been developed to provide the thermal science community with a collection of publicly accessible remote functions for performing complex combustion computation. This work represents the first effort to provide such an infrastructure to deploy a remotely accessible software service that allows developers of thermodynamics and combustion software to perform complex, multiphase chemical equilibrium computation with relative ease. This service can be integrated into any numerical application.

A researcher engaged in the development of software for simulating a thermal-fluid process will likely have a need to repeatedly compute the distribution of species resulting from a combustion reaction in many locations of a computational domain. If the number of equilibrium computations is in the tens of thousands, using a serial approach whereby computations are performed in a loop construct would likely be unacceptably time-consuming. Using CHEQS, however, equilibrium computations can be performed in parallel by asynchronously invoking a remote operation within the researcher's own application. There is no need for the researcher to design his or her own equilibrium solver or locate a suitable third-party library. The CHEQS Web service can essentially be thought of as code that "plugs in" to existing software and takes advantage of distributed computational resources over the Internet. This style of software development based on orchestrating loosely-coupled and distributed software services is called a Service Oriented Architecture (SOA). Adopting an SOA approach to building combustion applications will have a sweeping impact on research and teaching in the thermal sciences as developers are able to construct new software tools that build upon an ever expanding collection of independent and modular Web Services.

We have developed a chemical equilibrium Web Service that exposes an operation to calculate and return the equilibrium distribution of the products of an arbitrary reaction at a defined temperature and pressure (see fig. 1). The client application connects to the SOAP server and the temperature, pressure, reactants and products are passed as arguments. The input parameters are the reaction temperature in Kelvin, pressure in kilopascals, a comma and colon delimited list of reactants, and a comma delimited list of allowable products.

The list of reactant species is specified using the format [*moles*: *formula*]where *formula* is the chemical formula of a reactant species using the Hill naming system and *moles* is the quantity of the respective species in the reactant mixture. To illustrate how the Web Service can be used, consider the standard reaction for ammonia synthesis by means of the well known Haber process,

$$N_2(g) + 3H_2(g) \longleftrightarrow 2NH_3(g). \quad (6)$$

The Haber process is carried out at about 520°C and 500 atm in the presence of an iron-molybdenum catalyst. The catalyst increases the rate of the reaction but does not affect the reaction stoichiometry. At equilibrium, the mole fractions of nitrogen, hydrogen, and ammonia are approximately 17%, 50%, and 33%.

The CHEQS' Web Service solve operation returns the unique distribution of product species that corresponds to a reaction at a fixed temperature T in Kelvin and pressure p in kilopascals. To invoke the solve operation, a SOAP message body is constructed by a client process that includes a definition of four required input parameters for T, P, the reactants list, and the products list. The data is returned using simple JSON (JavaScript Object Notation, http://www.json.org) format, which makes it easy for clients to parse results.

2.3 Distributed Equilibrium Computation for Use in Fluid and Heat Transfer Simulation

Bhattacharjee and Paolini have developed the Flame3D [10] [11] application, which is based on the control-volume formulation of Patankar and Spalding. The goal of this coupling was to provide Flame3D with an ability to simultaneously compute, using the equilibrium Web Service, the equilibrium distribution of every control volume after a steady-state temperature field was found.

Figure 2 shows a snapshot of a slug flow simulation from Flame3D where carbon dioxide (CO_2) passes at a constant velocity u_x=49.212 mm/s over a semi-infinite hot plate configured at 6000K. The boundary conditions of all other walls of the physical domain are configured to have an ambient temperature of $T_\infty = 300\text{K}$. As CO_2 passes over the plate, some dissociation to carbon monoxide $(CO)$and oxygen (O_2) will occur. Once the distribution of species within each control volume is found, the concentration profile along with the temperature profile can be plotted to evaluate the respective concentration of each product as well as the thermal boundary layer.

Figure 2 shows a plot of the resulting equilibrium concentration of each product species and the thermal boundary layer in the fourth plot of the figure. Theta Θ is a dimensionless temperature given by

$$\Theta = (T - T_\infty)/(T_w - T_\infty) \quad (7)$$

where $T_\infty = 300\text{K}$ and $T_w = 6000\text{K}$. The red line is the expected theoretical thermal boundary layer with distance $\delta_t = 75.1700$ $\delta_t = 75.1700$ mm at the right wall and the interspersed green dots are numerically calculated values of δ_{tx} where $1 - \Theta = 0.99$ or, equivalently, when

$$(T - T_\infty)/(T_w - T_\infty) = 0.99 \quad (8)$$

is satisfied where $T = 357\text{K}$.

As one can see from the figure, the numerically calculated values of δ_{tx} closely coincide with the expected

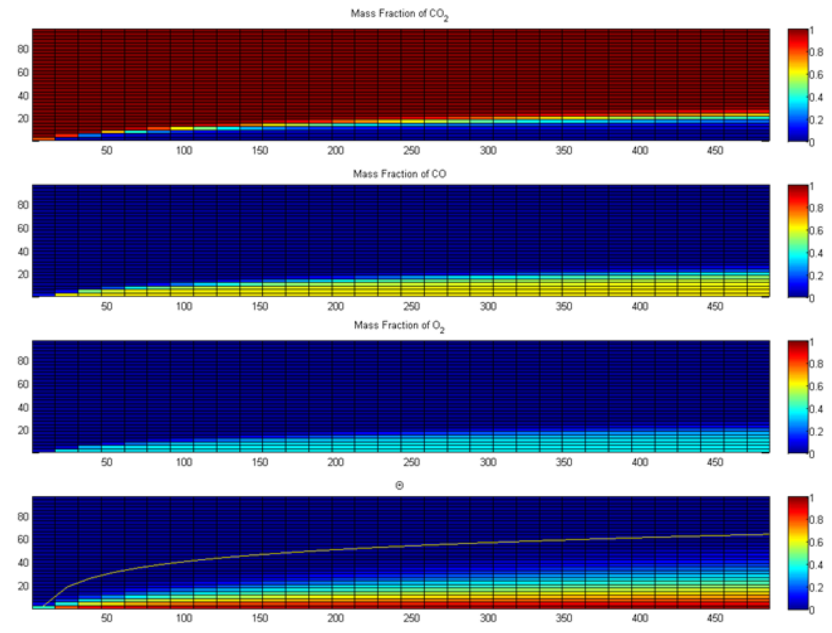


Figure 2. Concentration and thermal boundary layers of a steady-state ("slug") flow of carbon dioxide (CO_2) over a semi-infinite flat plate at 6000K. Theta Θ is dimensionless temperature and the yellow line is the thermal boundary layer with distance $\delta_t = 70.3125\text{ mm}$ at the right wall. [10]

theoretical δ_t using the approximation

$$\delta_{t,theoretical} = 4.92\left(x/\sqrt{Pe_x}\right) = 4.92\sqrt{\alpha x/u} \tag{9}$$

The thermal diffusivity parameter $\alpha = \left(k/\rho c_p\right)\ \left[\text{m}^2/\text{s}\right]$ was approximated as 2.2x10^{-5} which is valid for carbon dioxide gas at 1 atm and 300 K. PE_x is the dimensionless Péclet number with respect to the x-axis and is given by $Pe_x = ux/\alpha$. The Péclet number characterizes a flow's rate of advection to its rate of thermal diffusion. In the third plot of Figure X6, the red line shows the expected theoretical concentration boundary layer based on the approximation

$$\delta_{c,theoretical} = \delta_{t,theoretical}/\sqrt{Le} \tag{10}$$

and the green dots show numerically calculated values of $\delta_{c,x}$ where

$$\left(C_{O_2,S} - C_{O_2}\right)/C_{O_2,S} = 0.99, \tag{11}$$

$C_{O_2,S}$ being the concentration of oxygen gas at the surface of the plate. From the plot, one can see a disparity exists between the numerical and theoretical solution and this is due to the fact that the numerical solution is an *equilibrium solution* and does not account for the diffusion of oxygen that would actually occur in such a reaction. The dimensionless parameter in equation (10) is the Lewis number, which is the ratio of thermal diffusivity to mass diffusivity D_{AB} and is given by $Le = \alpha/D_{AB}$.

3 Many Task Computing Approach

3.1 Requirements

The Flame3D application computes the temporal and spatial evoloution of species within a volume. The volume is broken up into a grid of control volumes (CV's), each of which is used to compute the chemical composition and temperature of the species within the CV using the CHEQS chemical equilibrium Web service. Each CV calculation is independent of any other CV. As mentioned above, the application compute cycles are run over 1024 CV's until thermal equilibrium occurs (typically less than a few thousand iterations for 10^3 CV's). For high-resolution computations, CHEQS equilibrium services are designed to run using 10^6 CV's, over 5x10^5 iterations, for a total of 5x10^{11} total calculations. With the current design, each CV calculation takes about 1 second, resulting in a total compute time of thousands of years for large scale computations.

Based on these characteristics, we chose to focus on a multi-task computing (MTC) approach. Additional requirements include: the need for the service to operate within the heterogeneous computing environments typically encountered at a research university (local clusters and remote HPC systems, various policies and accounts); to interface to different queing systems; to be hosted as a Web service callable by Flame3D applications. The service also needs to support two types of clients: rapid response (results be returned rapidly to facilitate real time interactive computations for some applications); and large multi task computations (e.g. 10^6 or more CV's).

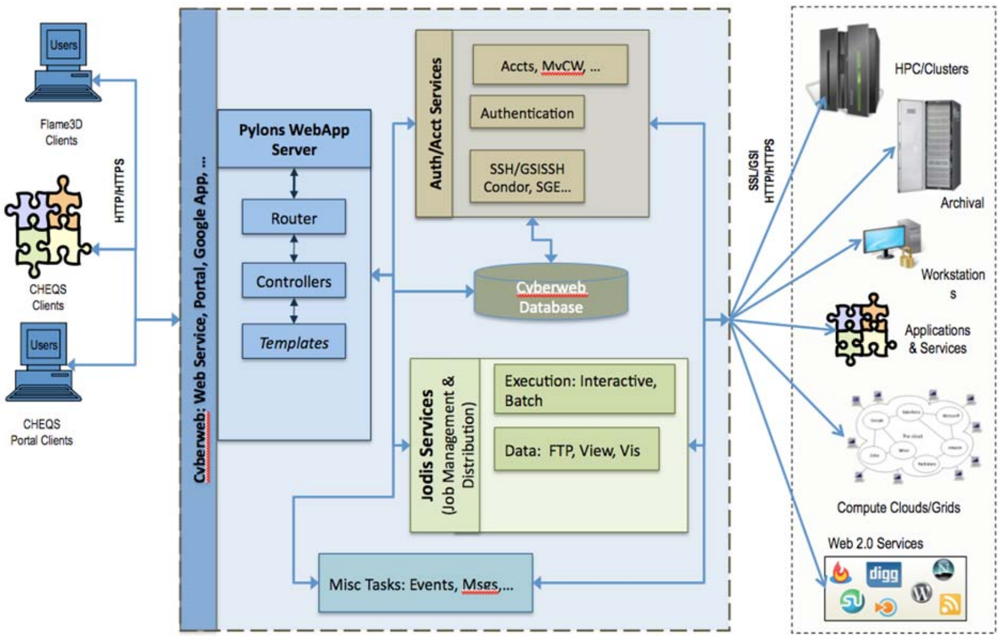


Figure 3. The JODIS and CyberWeb architecture. The diagram shows the client interface layer (left), the middle layer based on cyberWeb, and the back-end remote resources layer [12] [4].

To meet these requirements, the job distribution Web service framework (JODIS) was designed to distribute application workloads across heterogeneous computing systems. JODIS is a Web Service application framework based on the master/worker design pattern and utilizes the SDSU Cyberinfrastructure Web Application Toolkit (cyberWeb) as a hosting environment [3] [4].

3.2 SDSU Cyberinfrastructure Web Service Framework (CyberWeb)

JODIS services are hosted on the Web through the SDSU Pylons-based Cyberinfrastructure Web Service Framework (CyberWeb). This framework is based on the Pylons Web Application Framework [4], and includes components that interface with the distributed, services oriented architecture of the Web, as well as the cyberinfrastructure and middleware services such as those hosted on the TeraGrid (http://www.teragrid.org). CyberWeb can be used to host any type of Web 2.0 service, and can route URI requests, process requests and execute tasks [12]. Using cyberWeb, the JODIS system can host multiple services using a variety of interfaces.

One aspect of this project is based on an attempt to keep systems like JODIS simple. Python is the primary programming language for several reasons: it has a strong, active open source developer community; it is object-oriented; libraries have been proven to work well for science applications (SciPy, NumPy, PyNGL PyNIO); and libraries for grid integration (pyGlobus, pyGridWare, pyGSI). The cyberWeb framework must have the ability to integrate emerging Web 2.0 technologies including WSGI (Web Server Gateway Interface), databases, XML/JavaScript/AJAX, Google Gadgets, social networks, Web 2.0 services and toolkits, and security. Pylons is a Python based Web application framework, hence, any number of components and libraries developed by other software projects can be incorporated into the system, which facilitates customization. In the Pylons architecture, the services layer is decoupled from the logic of the code behind it, JODIS is hosted as a computational service that can be accessed by a portal, application, or other Web Services. A feature of Pylons is that all software can be bundled into a deployable egg that can be installed on any system and can be modified for use by other applications. System admin pages have been developed to facilitate configuration and management of users and resources. Thus, the JODIS system can be securely deployed across many hosting systems. The cyberWeb framework integrates with grid security infrastructure (GSI) authentication and the MyProxy credential services.

A feature of CyberWeb is that it has most of its configuration data stored within a relational database (users, hosts, authentication, jobs, job history) and all of these tables are administered using either the admin portal or a command line interface. Currently, CyberWeb uses SQLITE3, which is stored in memory within the server. Data is initialized using a flat text file which is in JSON format, which allows a new project to easily seed the database. During run time, all services, including JODIS, have access to this database. In this way, new resources can be added or modified and JODIS to control access to the service (if needed) using the authentication modules.

The CyberWeb framework provides us with a rapid development cycle and the ability to add new and improved RPC protocols in the future. Using cyberWeb to host services (and access the same code) means that a developer can develop one service (such as job submission via JODIS) and expose that service to different clients such as a portal, a remote application, or a Google gadget.

3.3 Job Distribution Services

JODIS consists of multiple services offered through a Web 2.0 application server environment. In the web application environment, all services work together to offer an end-to-end job dispatching service to multiple clients. The architecture for the JODIS system can be found in Fig. 2. The Web server environment (based on CyberWeb, see below) provides users with methods of communicating with JODIS using either a Python client API or Web Service to access the Job Service. The Web Service allows a wide variety of applications to interact with JODIS regardless of location, device or programming language. The Job Service provides a majority of the user-accessible function calls and manages a user's jobs regardless of the resource. The Resource Service works on the back end to manage the various connections between the JODIS and the compute resources being used. It works as a singleton to minimize the number of duplicate connections. The connections between JODIS services are shown in the architecture diagram (Fig.2). The ability for JODIS to gather usage information and use this data for predicting job runtimes and for selecting where to run a job provides a useful approach to running MTC jobs.

3.3.1 Job Services

Clients primarily interact with the JODIS Job Service API. It is responsible for tying all of the services together to provide simple job submission to heterogeneous resources. This service is used for job submission, query, and cancellation. This service hides many of the complexities involved with job submission such as choosing which resources to use, tracking these jobs, and managing connections between the resources. JODIS uses a job runtime "Guesstimation" along with a Distribution Policy to dynamically choose which compute resource to use or each job.

Job "Guesstimation": This capability provides JODIS with the ability to forecast job runtimes in order to choose the appropriate compute resource for each job. Many cluster queuing systems set maximum run time limitations on jobs to improve general availability of their resources to a wide audience. This limit is arbitrarily set by the system administrators of the cluster, and in several cases this limit was set as low as 24 hours (see Table 1). JODIS Job runtimes can be set via the user or based off of historical runtimes when JODIS is integrated with a database. Within reasonable limits, forecasts compute worst case estimates based on upper bound run time limits.

The JODIS Client Service dispatches a sample application to measure the expected runtime. CyberCHEQS jobs increase linearly with respect to the number of tasks. In our experiments, CyberCHEQS guesstimates come very close to the actual runtime. In the future, users will be able to estimate runtimes via historical data stored in a database and adjust these runtimes based on some sort of job size parameter.

Job Distribution Policy: This component takes in an array of guesstimated run times as its sole parameter. Using these guesstimates for each compute resource, JODIS makes an attempt to minimize the runtime of all the jobs given the restrictions on queue time limits and memory restrictions of compute resources. JODIS stores the queue time limits and memory restrictions in a configuration file loaded in at runtime. As these limits and restrictions change often, future versions will hopefully mine for this data when possible. JODIS does not make an attempt to estimate or take into account queue wait times. This is not an ideal, however, job forecasting is out of the scope of this project. To compensate for this, each resource can be given a weight factor. This weight is multiplied by the guesstimated runtime implying that a smaller weight factor improves the guesstimated time for a particular resource whereas a larger weight factor worsens the guesstimated time. One can imply that a factor of 0 across all resources will spread the tasks across those resources evenly. This becomes useful when differentiating between internal clusters versus the TeraGrid. The job queues of the TeraGrid were typically longer than those of our in-house compute clusters. Possible explanations include network issues, node configuration and job start up/node warmup times.

3.3.2 Resource Services

The resource service provides two essential functions to JODIS. The first being connections from JODIS to its compute resources. Secondly, the resource service wraps the functionality of batch queuing systems. The resource service offers communication to the compute resources and client targets mainly via Secure Shell (SSH) and GSI-Enabled SSH. The latter is used for the TeraGrid and other GSI-Enabled resources. The abstraction of these essential functions allows JODIS to easily extend its operations to

new protocols as they emerge. The resource service is a singleton class shared across all JODIS instances running on a particular machine in order to reduce useless duplicate connections. These connections can easily become out of control if, for example, a user creates 20 JODIS job instances, which in turn might create separate connections for each compute resource, quickly bogging down system resources on the client and server.

The resource service also wraps the functionality of batch queuing systems. Batch queuing systems are essential in maximizing CPU hours on compute clusters. They work to ensure jobs on all compute nodes are continually being exercised. JODIS relies on these queuing systems to execute jobs on each cluster. JODIS is able to use many popular queuing systems such as the Sun Grid Engine (SGE), Portable Batch System (PBS) and Condor. These systems provide the same core functionality each with its own slight syntactical nuances. Wrapping the basic queuing functionality and communication provides JODIS the flexibility to work with any queuing system over any communication channel offered by the resource service.

3.3.3 *Client Services*

JODIS hosts a general client Web Service for authorized job submission. Clients interact with JODIS directly using the Python API, or more popularly, through the SOAP Web Service interface. Both interfaces offer the same functionality, however, the Web Service interface gives clients the flexibility of running their code anywhere and offloading heavy computations to CI resources. The JODIS WSDL allows new users to find the service as well as keep our users up-to-date on the latest API. Developers can extended the client Web service for specific applications with the use of the Job Builder Client interface which exchanges messages with JODIS. The client service can hook into the JODIS job service to provide functions such as pre-processing, post-processing and more. The CyberCHEQS client service is specific to the CHEQS application. This layer is responsible for CHEQS specific tasks such as runtime guesstimations and job parsing. For CyberCHEQS, the total number of CVs (also called tasks) are distributed across the number of jobs to be run in the computation. The client is responsible for clustering these tasks into groups which are then passed as a collection of jobs to the JODIS job service.

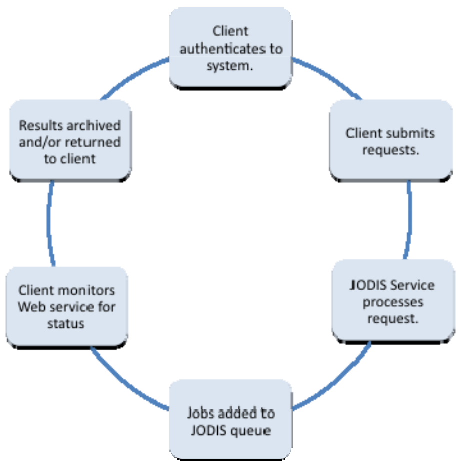

Client authenticates to system.
Client submits requests.
JODIS Service processes request.
Jobs added to JODIS queue
Client monitors Web service for status
Results archived and/or returned to client

Figure 4. Diagram of a typical JODIS job cycle for job submission from a CHEQS client [3].

3.3.4 *The Jodis Job Cycle*

A typical JODIS job cycle is shown in fig. 4 and helps to demonstrate how the various components interact, and is describe below:

- A client logs into and starts a JODIS session.
- The client adds jobs to be to run as the job set. The client tells JODIS when all the jobs have been submitted. This signals to JODIS to start its work.
- JODIS processes the jobs and collects metadata are collected by JODIS and added to the client's job queue. In the current implementation, each connection creates its own instance of JODIS to differentiate between job sets. JODIS guesstimates the job runtime and dynamically chooses where to distribute the jobs.
- JODIS adds jobs to the queues on the remote resources. JODIS guesstimates the job runtime and dynamically chooses where to distribute the jobs.
- The client can periodically check the status of running jobs. The client does not need to be concerned with the exact location of the job. JODIS stores this information but is typically abstracted from the user. As far as the user is concerned, each job is being run on the same resource.
- When the jobs are finished, JODIS delivers results directly to a specified file location or return the results to the user via a Web Service response. Actions can also be triggered such as sending an email or updating a database table.

3.4 Related Work

The field of MTC is new, and hence, the number of tools dealing with this topic is small. Even smaller is the infrastructure optimized specifically for MTC problems. The MTC community is developing a solid platform starting with a series of workshops and BOFs held at various conferences and meetings [13] [14]. During these workshops, several well-known systems were shown that explored similar topics – Pegasus, Falkon and Swift. While each of these frameworks exists in the same MTC problem space, it is difficult to compare these systems to that of JODIS since the problem requirements for each are different.

Pegasus is the most similar to JODIS. It is a framework that is designed to map workflows to the Grid environment. It is responsible for finding the resources on the Grid that are capable of performing the computations, the data used in the workflow and the necessary software [15] [16]. Pegasus was released in 2003 as part of the GriPhyN Virtual Data Toolkit. Falkon, a fast and lightweight task execution framework, designed by Raicu et al. enables users to execute a large number of small and rapid tasks. Their aim was to show that MTC problems could be run on very large scale HTC hardware. To accomplish this task, Falkon separates the allocation of resources using the conventional scheduler from the dispatching of small tasks. [17] Falkon has been optimized even further using its specially designed TCP protocol for communication between its client and compute nodes. Swift, based on CoG Karajan and Falkon, delivers a useful job management system using a new scripting language called SwiftScript. This tool provides users a useful tool to manage large datasets and managing all of the processing scripts across GSI-enabled resources. [18].

JODIS differs slightly from the above projects in that it is intended to be lightweight and adaptable, to run on open scientific computing infrastructure (e.g. TeraGrid, local clusters), to use standard applications where possible, and to use existing queues and policies. JODIS does not include a workflow definition layer. Instead, JODIS uses a simple Web Service API for batch job processing.new protocols as they emerge. Other types of job scheduling and distributed computing systems exist such as Condor [19], Globus, Sun Grid Engine, but they are not universally deployed. Instead, we intend to extend JODIS interfaces and clients to access these services.

TABLE 1.
Hostnames and Performance Data

HOST	CPUS USED	SERIAL TIME PER TASK	PAR TIME PER TASK	RAM	PROC.	QUEUE	LIMITS
dolphin.sdsu.edu	88	0.664	0.096	8 GB	Intel Quad Core	PBS	None
anthill.sdsu.edu	60	0.729	0.008	16 GB	Intel Quad Core	SGE	None
blackbox.sdsu.edu	60	0.529	0.200	12 GB	Intel Quad Core	Condor	None
tg-login.ncsa.teragrid.org (mercury)	128	2.78	0.035	4.47 TB	Intel® Itanium	PBS	24 hr; 128 node maximum
tg-login.tacc.teragrid.org (lonestar)	5,840	0.64	-	8GB	Dell Power Edge	LSF	48 hr; 40 Job max; 512 nodes max

4 EXPERIMENTS AND RESULTS

To measure the performance of the CHEQS applications, two types of tests were conducted: (a) performance tests of how well JODIS can distribute CHEQS equilibrium computations across resources; (b) integration of the JODIS services into and existing CHEQS based application. These tests were run on several computing resources including the TeraGrid and local private clusters running queuing systems including SGE, PBS, LSF, and Condor. The machines are described in Table 1.

4.1 Distribution of the CHEQS Equilibrium Computations

A CyberCHEQS client was developed to submit simple equilibrium test cases to JODIS, which distributed jobs across the compute resources. The purpose of these tests is to measure the scaleability of the JODIS plus equilibrium services system. In order to isolate latencies on the client side, the client was run on the same machine as the JODIS service. A job consists of a copy of the CHEQS equilibrium solver Java code and the reactant data for a number of CV's. The job returns the resultant products. This is a small test case: it computes results for 1 reactant and 3 possible product species. Two modes of operation were tested to simulate the most common use cases of CHEQS for the Flame3D application.

The first mode (referred to as 'single job') simulates the current methodology implemented in Flame3D, where each CV sends its own asynchronous request to a Web Service to update its own equilibrium composition. Hence, using this methodology in a volume with 100 control volumes, the CHEQS web server would receive 100 individual requests. These tests effectively defined the serial cases. The second method is the parallel (or ensemble) method, where multiple CV computations and data sets are batched together and sent to the Web Service as 1 request; and multiple requests are sent to the job service.

Tests were performed on a variety of hosts (see Table 1) and number of nodes (ranging from 1 to 128) using a single resource or in multi-mode (across multiple clusters). Scaling tests were performed as a function of both the number of nodes and for various numbers of CVs (1 to 10^7). The number of nodes utilized was a function of number of jobs and CV's, cluster size, queuing policies (which limited the number of jobs that could be submitted or how long a single job could last) and memory. In this case, without special permissions, large runs were not possible (e.g hundreds of MTC jobs).

Runtime and memory were profiled using commodity tools. Memory was monitored from within the Cyber-CHEQS application using the standard Java Runtime library. The memory was profiled during key points during the application run. For most cases, the maximum heap memory usage was measured.

Runtimes were a little more complicated to measure. Runtime results are reported the longest runtime across all nodes + the time to transfer result files. For reference, preliminary published results of CHEQS running Flame3D using Condor for 1, 2, and 4 nodes [2] on a control volume grid of 1024 cells are used for comparison.

4.1.1 Results

Figure 5 shows the log of the total runtime vs. the log of the total number of nodes, as a function of the number of processors for the SDSU anthill cluster. As expected, the runtime linearly increases with the number of tasks increases and the runtime decreases as the number of nodes increases. This type of result was observed for all systems used in this study as well as for runs involving heterogeneous collections of nodes. The curve marked “Blackbox-Projected” is a projection based on tests run on the CHEQS system shown in fig. 1, using a 1, 2, and 4 node Condor system, for ensemble runs of 1000 control volumes. In these cases, the best time per task was 0.200 seconds.

The average time per task is a key factor in determining how JODIS will distribute tasks across a pool of nodes. It is important to distinguish between the per-node run time for a task and the effective parallel runtime for an ensemble of jobs. For larger jobs (ignoring small jobs where warmup times dmoniate), the time per task for the average parallel time per task is given by $T_{par} = T_{to}t/T_{tasks}$. Figure 6 shows the time per task for the different systems performance when running the ensemble jobs in parallel, and Table 1 contains a summary of average times for all machines.

The per-node/per-task runtimes of 2.56 seconds for mercury (NCSA) are anomolies. It is not clear why the performance of on mercury is worse than the other machines. Possible explanations include using an un-optimized installation of Java or the GSISSH library used

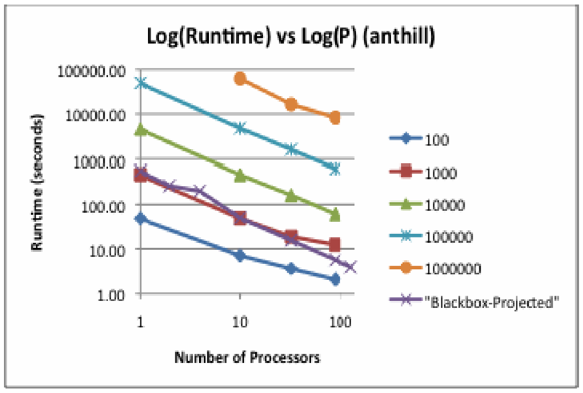


Figure 5. Log of the runtime vs the log of the number of nodes as a fundtion of the number of tasks on anthill.

is not as fast as the SSH version. Note that for smaller numbers of jobs the time per task fluctuates and should be taken into account when distributing the workloads.

The speedup vs. the number of processors on anthill, as a function of the number of tasks, is shown in fig. 6. Speedup is given by Sp=Tser/Tpar. The speedup on anthill begins to converge for large jobs for the larger processor runs; but there is still speedup occurring and this trend should continue for larger jobs as long as the node memory is not maxed out. Speedup on mercury, for the larger number of tasks, is near linear speedup, with no signs of leveling out as the number of processors increases. Due to queuing limits, the maximimum number of jobs was limited to 128.

Overall, the performance improvements are satisfactory. For the case of the initial 4 node Codor runs, the total run time was 200 seconds for 1000 CV. Using JODIS, running 1000 tasks on 4 nodes, the total time would be around 130 seconds, for a time per task of 0.13. This represents a reduction in runtime by a factor of about 50%. However, actual CHEQS calculations are done for more complex reactions and this could make the JODIS job run longer, but nonetheless the results are encouraging. Note that this job can be run on anthill in less than 10 seconds. From another perspective, the JODIS system could run up to 50,000 control volume calculations in about 650 seconds. This provides CHEQS with the ability to increase resolution without adding significant return times to solution from within its existing web system.

Another factor affecting performance and job distribution is the memory used by a large ensemble job. The size of a single control volume is small, approximately 100 bytes (depending on the size of the reactions), but for larger jobs this can be significant. For example for 10^5 CV’s, the request packet will be about 10 Mbytes. JODIS measures the memory usage as a function of number of CV's, number of nodes, architecture/host. This data is stored and used to predict job requirements for matching to resources. We find that the CHEQS jobs fall into two memory usage categories: for small numbers of tasks ($<10^3$) the memory usage is dominated by the size of the VM (about 0.6 MB on anthill and mercury); while for

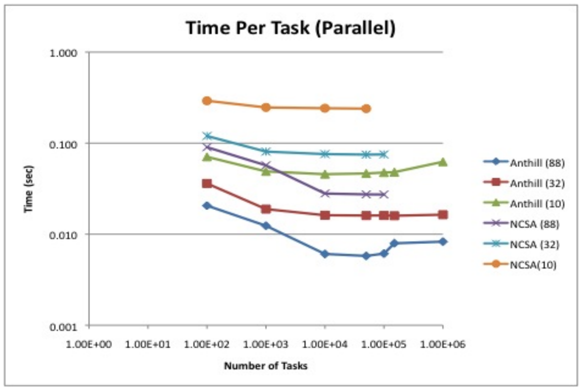


Figure 6. Average runtime per task for the ensemble of tasks run in parallel as a function of nodes.

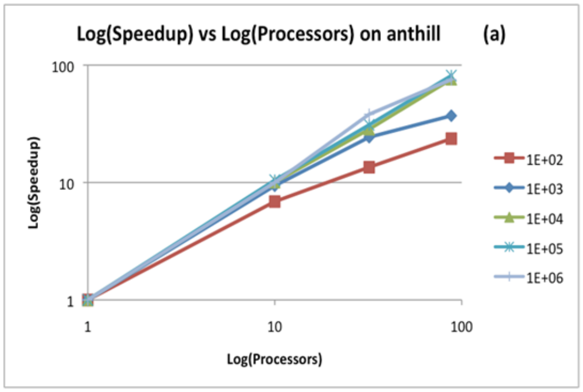


Figure 7. Speedup vs number of processors as the total number of tasks increases for the SDSU anthill cluster.

larger jobs (10^3 to 10^6 CV's) the memory averages around 0.250 and 0.23 Bytes/task on anthill and mercury, respectively.

The JODIS system is also capable of distributing tasks across heterogeneous resources. This was tested using anthill, blackbox, TACC and NCSA clusters for various test conditions. This is discussed in the next section.

4.2 Integration of JODIS and the Flame3D Simulation Application

Once the JODIS services were characterized and validated, we then moved on to inserting JODIS into a real CHEQS application. In particular, we worked with the Flame3D simulations (described above in Section 2.3). The JODIS service replaces the Web service located on right hand side of fig2. With JODIS, the CHEQS equilibrium solver is sent to the compute nodes along with the equilibrium data. We wanted to also test a more realistic scenario, so the simulation computes the dissociation of air flowing over a hot surface:

$$O_2 + 3.73N_2 + 4.44\times10^{-02} Ar + 1.81\times10^{-03} CO_2 \square \left\{ \begin{array}{l} \mathrm{CO}_2,\ CO,\ O_2,\ N_2,\ O_3,\ NO,\ N_2O,\ NO_2,\ O,\ N, \\ Ar,\ C(g) \end{array} \right\} \quad (12)$$

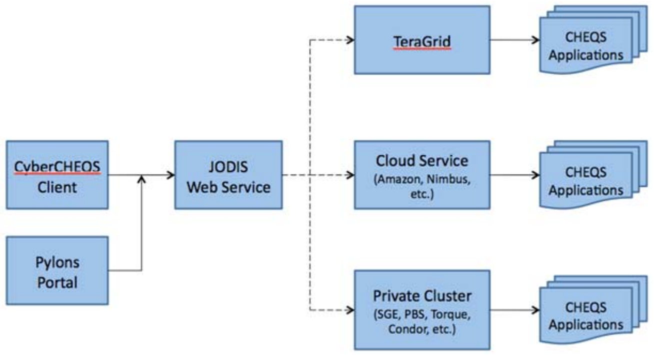
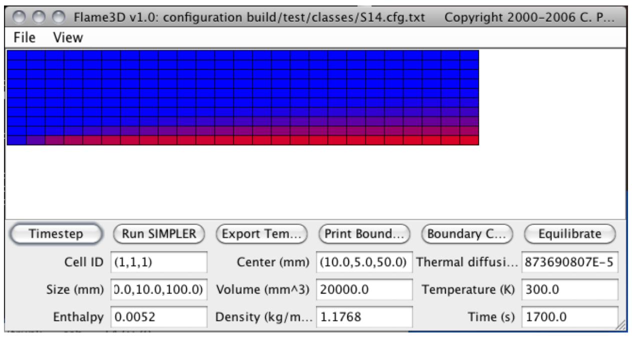


Figure 8. Diagram showing the JODIS enabled CyberCHEQS architecture [3].



Flame3D v1.0: configuration build/test/classes/S14.cfg.txt Copyright 2000–2006 C. P...

Cell ID	(1,1,1)	Center (mm)	(10.0,5.0,50.0)	Thermal diffusi...	873690807E-5
Size (mm)	0.0,10.0,100.0)	Volume (mm^3)	20000.0	Temperature (K)	300.0
Enthalpy	0.0052	Density (kg/m...	1.1768	Time (s)	1700.0

Figure 9. Snapshot of the Flame3D GUI [2] showing control buttons for temperature, timestep, etc. The plot shows the temperature distribution across the plate for a 10x20 matrix of control volumes used in these eperiments.

The GUI client is written in Java, so the client can run anywhere (see figure 9). The Flame3D simulations are run manually from within a GUI application running on our laptops or local machines. The JODIS service is running on a workstation at SDSU.

The goals of the experiment were to solve technical challenges required to insert JODIS as the Web service used by the application; to see if we could run larger, more complex simulations and to increase the number of control volumes (typically 10^3 max). Each CV task is small (about 100 KB/task), but in aggregate numbers the actual data can be significant: for 10^5 CV's, the data is on the order of 10 MB. In the existing mode, the application sends one Web service request at a time, which results in long delays as a result of the job submission times. In the initial design, the client bundles up batches of tasks and sends them in groups to the job service.

Using this simple mechanism, we were able to run tests for cases up to 10^5 CV's using nodes across 3 clusters. The average time/task was about 0.1 seconds. However, we found that the application hangs for these large jobs, and will require further modifications. Figure 10 shows the computed temperature profiles for this case, which is similar to those of fig. 2, which verifies that the system could reproduce results. Further testing will include modifying the code to run in a more automatic mode, modification of the graphics display to handle the large number of CV's and reconfigureing the timeout parameters.

5 Conclusions

Although there exist more complex queuing systems (such as Condor, Falkon), these are not always available or easily installed. Hence the motivation to build a simple system that uses MTC methods. Simple in design, JODIS has significantly improved the CHEQS performance overall by providing the ability to run larger simulations (more control volumes), in parallel, with faster times to results. JODIS can submit jobs to any queuing system, so it can interface to these larger systems. JODIS is also flexible: it is being used by other applications and for jobs run within CyberWeb such as third party file transfer.

There are several technical challenges that we encountered (and mentioned above) that limited what could be accomplished, especially when attempting to run large numbers of jobs on large-scale machines, such as those on the TeraGrid. While these conditions may be common, or even required for the current infrastructure configurations. These are summarized below:

Queuing Limits: These have a significant impact on the tests we were running. This was mostly due to the queuing limits established by the TeraGrid compute resources; most likely because these resources are historically are weighted towards massively parallel computing problems. On the NCSA Mercury cluster, tests were limited to running a maximum of 128 jobs within a 24-hour period. CHEQS jobs are run as ensembles, so 128 nodes would be acceptable, but this limits the number of tasks we can run. Other queues allowed more time but fewer processors.

While the maximum number of jobs is set to 128, the maximum number of processors allowed is higher than 128. This works well for MPI-based applications, where 1 job maps to P processors. However, since CyberCHEQS has a 1:1 mapping of jobs to processors, we were unable to utilize all 640+ processors It is not within the scope of JODIS to parallelize an application, but to simply to distribute the work. Other MPI-enabled applications will hopefully be able to use JODIS to distribute jobs across the entire cluster.

The implication of this issue is that MTC problems require new approaches to queuing and scheduling if we are able to take advantage of these larger systems. Additionally, although one could send a request to reserve a large number of nodes, this does not map well to real-time simulations that the CHEQS application will be routinely running.

Universal Resource Availability/Queuing Estimates: if MTC systems are going to be successful, we need to have a way to identify what resources are available and

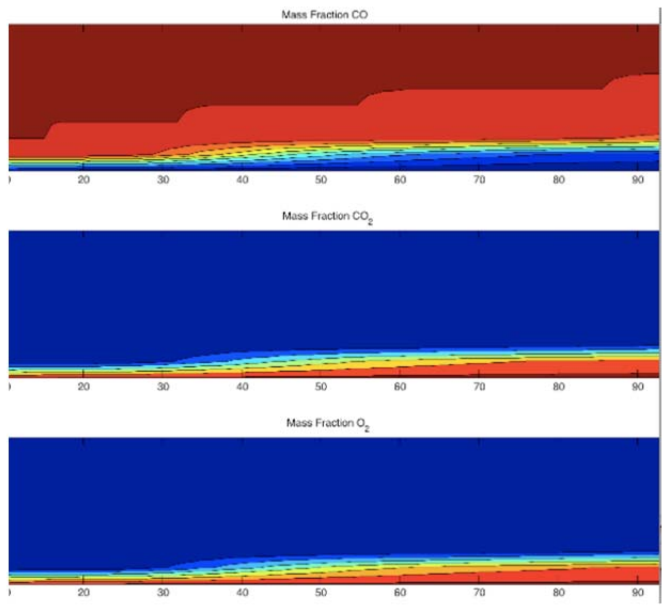


Figure 10. Concentration and thermal boundary layers of a steady-state ("slug") flow of carbon dioxide (CO_2) over a semi-infinite flat plate at 6000K. Theta Θ is dimensionless temperature and the yellow line is the thermal boundary layer with distance $\delta_t = 70.3125\,\text{mm}$ at the right wall [5].

when, and for how long. Ideally, these services will allow MTC jobs to be used as backfill jobs.

Java Threads: We encountered an interesting phenomenon while running our experiments on several compute resources including the SDSU Dolphin cluster (whose results are not reported) and NCSA Mercury. It was noticed in several environments that multiple processes were created for each JVM instance that was run. The application ran with no errors. We would not have noticed this, except that our runs on Dolphin froze the compute node and in turn locked up the PBS queue (consequently annoying system administrators and other users). Note that while multiple processes were observed on both Dolphin and Mercury, the processes on Mercury did not lock up the PBS queue as observed on Dolphin. This is possibly a queuing configuration limit, but it not resolved at this point. The issue further intrigued us when we noticed Anthill running same operating system and the same version of Java showed different behaviors. Further research showed that these processes were various different Java threads, which agreed with the observation that all but one thread sit idly while one lone thread consumed CPU time. While these threads did not seem to consume resources, the full effect of these threads on a large CyberCHEQS simulations is unknown.

RPC Standards: Many different RPC protocol and standards exist out there today. A flexible form of communication between the client and the JODIS Web Service is needed. The Simple Object Access Protocol (SOAP) is a good choice, due to its flexibility and its integration with the Web Service Discovery Language (WSDL). The WSDL defines services offered by a Web application. These definitions make client integration with the Web Service very simple. Packages exist for most programming languages that create stubs for these RPC calls. Using these stubs abstracts the fact that the client is using any remote function at all, making integration with these services quick and easy.

The Globus Alliance and IBM created the Web Services Resource Framework (WSRF) in 2003. Normally, Web Services are considered to be stateless. A Web Service does not normally remember history. WSRF aimed to fix this by defining a new standard that provides the notion of states. Unfortunately, its requirement of WS-Addressing metadata made it incompatible with the standard SOAP protocol. The framework was controversial at the time and has since faded from the limelight. While WSRF is still in use today, it is not as popular as it once was. Since the majority of SOAP users use the standard protocol, many Web Service providers choose to implement persistent storage via a database rather than maintaining states in the Web Service. Our goal is to make JODIS widely available and based on commodity out-of-the-box software. It was decided for JODIS to stick with the standard SOAP protocol. The addition of states could later be added using the cyberWeb database.

6 Future Work

We will continue to modify JODIS in order to improve performance of the CHEQS service, and to increase its performance in higher resolution simulations and near real-time monitoring of the simulations. The project has the potential to impact hundreds of researchers and students as they use the CHEQS web portal interface while accessing high-end HTC resources on the TeraGrid.

JODIS is an on-going project whose focus will continue towards exploring more possibilities of integrating CI resources with new and emerging Web 2.0 technologies. Future work efforts include integration with the Cyberinfrastructure WebApp Toolkit (CyberWeb). A demo portal interface to JODIS job submission and monitoring tools via a portal will be developed. The JODIS application framework will be the default cyberWeb job submission mechanism. JODIS will utilize the cyberWeb database to complement its existing features, with common functionality, such as: (a) administering user access to the Web Service and gateway; (b) database capabilities for configuring resources, monitoring jobs and managing results, providing access to historical job data; (c) provide easy integration with other common RPC protocols. A goal of the CyberWeb project is that the software be widely adopted freely by scientists everywhere. The CyberWeb project achieves this goal through the use of Python eggs (which are similar to Makefile or Maven for Java) to automate the installation processes. These eggs take much of the worry out of installation and software dependencies. This capability will be used to generalize JODIS for download and use by other projects.

Another major goal for the JODIS project is scalability. Cloud computing offers a unique scaling opportunity allowing users to only pay for the computing power they use. Using cloud resources will allow JODIS yet another real-time mode of operation that can be used along with the current offering of HPC and HTC resources. A benefit of using cyberWeb and the installation egg is that this should facilitate easy migration of JODIS tasks to different cloud resources.

7 ACKNOWLEDGMENT

This research was supported in part by the National Science Foundation (Grants NSF-0753283, NSF-0721656), and with resources available on the NSF TeraGrid Project (NSF-12345, Texas Advanced Computing Center and the National Center for Supercomputing Applications), and NSF grant DBI-0850356 (anthill.sdsu.edu), and the San Diego State University Computational Sciences Research Center. (dolphin.sdsu.edu).

REFERENCES

[1] Bhattacharjee, S. and Paolini, C. P., Property Evaluation in The Expert System for Thermodynamics ("TEST") Web Application, Journal of Computer Coupling of Phase Diagrams and Thermochemistry - CALPHAD, 33(2), p 343-352, June 2009.

[2] Paolini, C. P. and Bhattacharjee, S., A Web Service Infrastructure for Distributed Chemical Equilibrium Computation, Proceedings of the 6th International Conference on Computational Heat and Mass Transfer (ICCHMT), May 18–21, 2009, Guangzhou, China, p. 413-418.

[3] The Job Distribution Service (JODIS) Project Website. Available: http://acel.sdsu.edu/JODIS

[4] Thomas, M. P. (2008), Using the Pylons Web Framework for Science Gateways. rid Computing Environments Workshop, GCE'08 12-16 Nov. 2008.

[5] Bhattacharjee, S. and Paolini, C. P., The Chemical Thermodynamic Module of The Expert System for Thermodynamics ("TEST") Web Application, 2009 ASEE Annual Conference & Exposition, June 14–17, 2009, Austin, TX

[6] Dong, X., Gilbert; K.E., Guha, R.; Heiland, R.; Kim, J.; Pierce, M.E.; Fox, G.C.; and Wild, D.J. Web Service Infrastructure for Chemoinformatics. J. Chem. Inf. Model., 2007, 47(4), 1303 – 1307.

[7] Truong, T.N.; Nayak, M.; Huynh, H.H.; Cook, T.; Mahajan, P.; Tran, L.T.; Bharath, J.; Jain, S.; Pham, H.B.; Boonyasiriwat, C.; Nguyen, N.; Andersen, E., Kim, Y.; Choe, S.; Choi, J.; Cheatham, T.E.; and Facelli, J.C.; Computational Science and Engineering Online (CSE-Online): A Cyber-Infrastructure for Scientific Computing, J. Chem. Inf. Model., 2006, (46), 3, 971 - 984.

[8] Frenklach, M.; Packard, A.; Seiler, P.; and Feeley, R. Collaborative Data Processing In Developing Predictive Models Of Complex Reaction Systems. Int. J. Chem. Kinetics, 2004, (36), 57 – 66.

[9] Goodwin, D. G.; CANTERA: An Open-Source, Object-Oriented Software Suite for Combustion, NSF Workshop on Cyber-based Combustion Science, National Science Foundation, NSF Headquarters, Arlington, VA, April 19-20, 2006.

[10] Paolini, C., Yeo, K. H. and Bhattacharjee, S., An object oriented formulation for the finite volume simpler algorithm. In Proceedings of the Western States Section/The Combustion Institute, October 2003.

[11] Paolini, C., Yeo, K. H. and Bhattacharjee, S., An object oriented formulation for unsteady 3d heat transfer. In Proceedings of CHT-04 ICHMT International Symposium on Advances in Computational Heat Transfer, April 2004.

[12] Thomas, M. P. and Castillo, J. E. (2009). Development of a Computational Environment for the General Curvilinear Ocean Model. To appear in the Proceedings of SciDAC 2009, June 2009, San Diego, CA, USA. Journal of Physics: Conference Series, 2009.

[13] I. Raicu, Z. Zhang, M. Wilde, I. Foster, P. Beckman, K. Iskra, and B. Clifford, "Toward loosely coupled programming on petascale systems," in SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing. Piscataway, NJ, USA: IEEE Press, 2008, pp. 1-12. [Online]. Available: http://dx.doi.org/10.1145/1413370.1413393.

[14] Workshop on Many-Task Computing on Grids and Supercomputers, 2009. MTAGS 2009. [Online]. Available: http://dsl.cs.uchicago.edu/MTAGS09/.

[15] Deelman, E., Blythe1, J., Gil, Y., Kesselman, C., Mehta. G., Patil1, S., Sua, M., Vahi, K., Livny M., "Pegasus: Mapping Scientific Workflows onto the Grid," Across Grids Conference 2004. [Online]. Available: http://pegasus.isi.edu/publications.php

[16] Deelmana, E., Singha, G., Sua, M., Blythea, J., Gila, Y., Kesselman, C., Mehta G., Vahi, K., Berriman, B., Good, J., Laity, A., C. Jacob, J., Katz, D, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," Scientific Programming Journal, Vol 13(3), 2005, Pages 219-237. [Online] Available: http://dev.globus.org/wiki/Incubator/Falkon

[17] Raicu, I., Zhao, Y., Dumitrescu, C., Foster, I., Wilde, M., "Falkon: a Fast and Light-weight tasK executiON framework," IEEE/ACM SuperComputing 2007. [Online]. Available: http://pegasus.isi.edu/publications.php

[18] Zhao, Y., Hategan, M., Clifford, B., Foster, I., Laszewski, G., Raicu, I., Stef-Praun, T., Wilde, M., "Swift: Fast, Reliable, Loosely Coupled Parallel Computation," in IEEE International Workshop on Scientific Workflows, 2007. [Online]. Available: http:// www.ci.uchicago.edu/swift/papers/.

[19] Tannenbaum, T., Wright, D., Miller, K. and Livny, M., Condor - A Distributed Job Scheduler, in Thomas Sterling, editor, Beowulf Cluster Computing with Linux, The MIT Press, 2002.