Experiments with Distributed Component Computing Across Grid Boundaries

Maciej Malawski*, Marian Bubak*†, Michal Placek*, Dawid Kurzyniec‡ and Vaidy Sunderam‡
*Institute of Computer Science, AGH, Krakow, Poland
Email: {malawski,bubak}@agh.edu.pl
† ACC CYFRONET-AGH, Krakow, Poland
‡ Dept. of Math and Computer Science, Emory University, Atlanta, GA, USA
Email: {dawidk,vss}@mathcs.emory.edu

***Abstract*— There are situations when a user has simultaneous access to computing resources in many virtual organizations built on different Grid infrastructures, as well as in local clusters or single machines outside Grids. Such a user may be interested in running a large-scale application on all available resources simultaneously, avoiding huge administrative overhead.**

We argue that in such situations the component model which can support parallel execution of distributed components may be very convenient, especially for scientific applications which go beyond pre-installed modules. We show how to exploit the dynamic runtime nature of CCA model, and define mechanisms for creating components with parametrized numbers of ports.

As a user-centric method of resource aggregation across Grid boundaries, we create a virtual pool of H2O kernels, spawned on the resources, using such protocols as SSH or batch scheduling and brokering systems. We also use a discovery mechanism based on HDNS and simple scheduling to allocate components.

We validate our approach by experiments, running a compute-intensive minimization application.

I. Motivation and research goals

Grid systems are designed for sharing and aggregating computing resources across administrative domains, by exploiting the idea of virtual organizations. There are situations, however, where a user has access to multiple virtual organizations built upon different Grid infrastructures, as well as to local clusters or single machines that are not part of Grids. Such a user may wish to run a large-scale application on all available resources simultaneously. This should be possible without the need for setting up a new virtual organization, which usually requires Grid middleware interoperability and imposes a huge administrative overhead. Instead, the user-centric scenario should allow for ad-hoc and transient collaborations formed spontaneously across Grid systems.

There are two key issues related to realization of this scenario. The first one is the selection and possible extension of an appropriate programming model, enabling flexible composition of distributed applications that can adapt to the heterogeneous environment. This issue is of great importance especially for scientific applications that go beyond pre-installed modules. We argue that the component model that can support parallel execution of distributed components may be very appropriate and convenient for programming such applications.

The second issue is to devise an efficient method for a user-centric aggregation of the computing power of such distributed and highly heterogeneous resources, which may have different access rules, application setup and execution mechanisms and may belong to many virtual organizations. The aggregation method should hide the complexity of the underlying infrastructure and provide a virtualized layer for executing the application.

In this paper we propose solutions of this problem which combine the programming model for distributed applications with methods for aggregation of available infrastructures to gather computational resources across the boundaries of Grid and cluster systems. The approach we describe is user-centric one and avoids the need for involvement and effort on the part of Grid system administrators. We validate the proposed approach with experiments on a compute-intensive minimization application, using the Common Component Architecture (CCA) [1] model implemented in our MOCCA [2] framework based on the H2O platform [3].

The paper is organized as follows. In Section II we provide an overview of related work in the area of components in the context of Grid infrastructures, and we briefly introduce H2O and MOCCA. Section III describes how the CCA model and our approach are used to enable parallel constructs therein. Section IV presents details of our method of user-centric resource aggregation and application deployment and Section V describes experiments with the real-world application. Conclusions and outline of future research are presented in Section VI.

II. Background

A. Programming model

Several programming models may be considered for distributed computing that transcends administrative and geographic domains. The first possible candidate is a distributed virtual machine based on the message-passing paradigm. The main drawback is the lack of support for environments whose properties change dynamically as is the case with Grids. Other alternatives, including Service Oriented Architecture, Grid, and Web Services, which are very popular nowadays and convenient for systems integration [4], have extremely high XML overheads in their use of WSDL and SOAP

documents, unclear vision of statefulness and standardization issues, and, most importantly – static binding between services and providers. This type of binding leads to difficulties for clients wishing to build custom applications which are more complex than merely invocation of available services installed by others.

Another (perhaps more appropriate) approach is the component model. Components, similarly to services, enable composition by separating interfaces from implementation. Moreover, they also provide natural mechanisms for remote component deployment, which is of key importance for such aggregated computing. There is a number of specific component models which have received much attention from the High Performance and Grid computing community [5]. In addition to specifying components and their interactions, these models also aim to support parallel programming constructs and to define mechanisms of components deployment on distributed resources.

One approach to parallelism in component programming is to consider components consisting of multiple processes, running on a high performance parallel machine or cluster. Processes within such a a *parallel component* may internally communicate using such mechanisms as e.g. MPI. This approach is being investigated in the CCA [1] model, taking into account such issues as data redistribution for MxN component connections [6]. A parallel extension to the CORBA Component Model (CCM) which also focuses on high performance is proposed within the GridCCM [7] activity.

Another type of component parallelism can be useful for more distributed and loosely-coupled scenarios, and appears as multiple components of the same type, often referred as *component collections*, running on distributed resources. An example of applying such approach to CCA model can be found in the master-worker application scheme implemented using the XCAT framework [8] or the DG-ADAJ system for Desktop Grids [9]. The ProActive implementation of the Fractal component model [10] offers mechanisms for hierarchical composition of parallel and distributed components, and also defines collective ports based on group communication. A skeleton-based approach can also be used to facilitate parallel constructs in component models, and is represented in such projects as ASSIST [11] and HOC-SA [12]. The importance of supporting parallel components is reflected in recent endeavours within the CoreGRID project for defining the Grid Component Model (GCM) [13], which is based on and extends the Fractal model.

Component models may define mechanisms for component deployment, e.g. CCA specification defines the *BuilderService* port, which is used for component instantiation and assembly. The implementation of this interface is left to framework developers; XCAT offers mechanisms for running components using the Globus Toolkit. CCM includes packaging and deployment in the specification, and there are solutions for deploying Corba components on the Grid infrastructure [14] using Globus. The ProActive system provides a mechanism for application deployment based on the concept of virtual nodes and deployment descriptors [15], and it supports various methods of running applications on resources where ProActive is installed, including most common cluster scheduling systems and Grid middleware. The Grid Component Model also addresses the deployment problem and proposes a common solution for automating this process [16].

B. Aggregation of computer resources

We can observe that great enthusiasm for building Grid systems has led to the development of many middleware solutions and a diversity of installations across countries. As examples we can point to such projects as the European DataGrid, CrossGrid, Unicore, GridLab and EGEE, the overview of which can be found e.g. in [17]. Unfortunately, even if the user has access to many of these systems, they are not interoperable, which makes it difficult to use their resources in an aggregated way.

There are several initiatives targeting Grid interoperability (e.g. projects such as GLUE [18] and UniGridS [19]), but their goal is first of all integration of middleware and infrastructure at the organizational level and they require additional work on commonality issues concerning security, information systems and policies. The unification of various cluster resource management systems was one of the initial motivations behind the development of Globus GRAM, and thus this solution is also organization-centric.

C. H2O and MOCCA

A promising alternative approach to resource sharing and virtualization is the H2O [3] project which introduces the concept of lightweight service containers called H2O kernels. The kernels allow authorized remote users to deploy and run their code, called pluglets. The deployed pluglets may be used by other parties, therefore the roles of resource providers and service providers, which are tightly coupled in traditional Grid systems, may be separated in H2O. H2O aims to support multiple programming models, among them MOCCA [2] which is our implementation of the CCA component model.

MOCCA, as a distributed CCA framework build on top of H2O, enables remote communication between components and facilitates the process of component deployment. For the communication, the RMI model is used, leveraging the extensible RMIX library. Components may be dynamically deployed on H2O kernels, and their code is automatically loaded from the location specified as a URL.

Our initial experiments with the MOCCA implementation show promising results regarding performance and flexibility, but the main drawback for running applications is the relatively low level of H2O adoption and lack of large-scale installations. Therefore, the possibility of using resources available with existing Grid installations becomes an important issue.

III. CCA Programming Model for Distributed Components

The first objective of our research was to choose and to validate the programming model which would be suitable for

running a custom, user-deployed application on heterogeneous resources. As we can see, a component model that supports parallel constructs and deployment mechanisms may be appropriate for such a scenario. Specifically, we have selected the CCA component standard implemented in MOCCA, and this decision was motivated by our experience with this framework.

A. Approach to parallel constructs

In parallel computing scenarios it is often necessary to simultaneously run multiple components of the same type across many distributed resources. In this paper we present our approach to introducing support for parallel constructs into the CCA model. Our strategy is based on exploiting the dynamic runtime nature of CCA components [8], which enables deferred definition of a component interface. In CCA, a component may add a *uses* or *provides* port during application execution, when required by runtime conditions. We use this feature to define components with a variable number of ports of a single type. Such *multiple uses* ports are suitable for constructing gather-scatter topologies of components. Another feature that we leverage is the connectivity between many users and a single *provides* port.

Fig. 1 shows an example of multiple components connected to a single *provides* port. The server component is not aware of the actual number of clients connected to its port and handles their invocations in the very same way as if they were coming from one component. When there is a need to connect a client component to multiple providers, the client must have one *uses* port for each provider. As CCA allows components to register the *uses* ports to the framework at runtime, their number doesn't have to be fixed – instead it can be parametrized or even dynamic.

We have defined a mechanism to facilitate such parametrized multi-port components. Following component instantiation, the number of ports can be set as a property using *ParameterPort*. Subsequently, the component registers its multiple ports with the framework, assigning them names which (by convention) end with consecutive numbers. Such multiple ports may be then connected, in a simple way, to multiple components. To enable this, we have extended the *BuilderService* interface to handle such multiple components and connections. It is worth noticing that such an approach can lead to future possibilities for more dynamic changes of port numbers at runtime, e.g. in response to changes in the infrastructure within which the application is running.

B. CCA model specific details

For application assembly CCA does not define the architecture description language (ADL) and therefore the mechanism for creating and connecting components is framework-specific, depending only on the *BuilderService* port defined in the specification. For this purpose our framework supports Java API or Python scripts which are flexible and can be easily edited when needed. The disadvantage of this approach is the lack of support for automatic tools, enabled by e.g. XML-based ADLs. Adoption of the ADL is currently on our research agenda in the context of CoreGRID GCM [13].

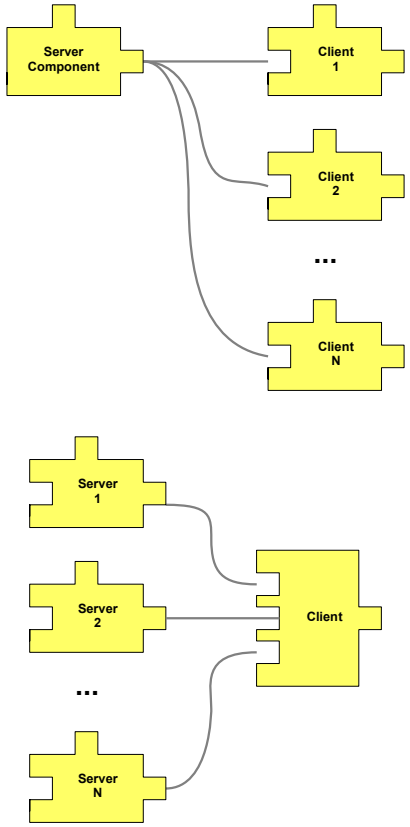

Fig. 1. Example of multiple ports and components. It is possible to connect multiple clients to a single *provides* port (top figure), whereas it is necessary to create multiple *uses* ports in a component, which is a client to multiple providers (bottom figure).

Unlike other models, CCA does not distinguish between functional and non-functional interfaces and all ports are treated in the same way. Separating non-functional interfaces may be useful in order to emphasize different aspects of components, as in the Fractal model. Nevertheless, CCA defines some "standard" ports, which may be useful for non-functional aspects. Specifically, the CCA specification defines the *ParameterPort*, which can be used to set the properties of a component, and currently we use it for introducing specific conventions to customize components. This doesn't mean changing the CCA specification – rather, we extend it with specific conventions.

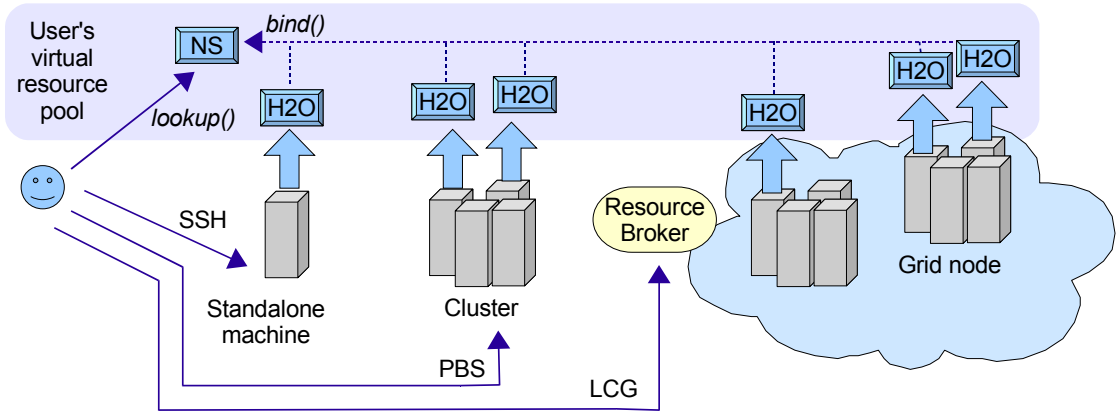

Fig. 2. Setting up the user's virtual resource pool. A user starts the HDNS server ("NS"), and then spawns H2O kernels on any machines that are accessible. SSH may be used for standalone machines, the PBS queueing system – for a local cluster and specific middleware, such as LCG with its Resource Broker – for Grid systems. Once the kernels are up, they are registered in NS and form the user's virtual pool of resources, available for deployment of the component application.

IV. Aggregation of Resources

Once we have selected the programming model, the second stage is the development of a method of aggregating computing resources for application deployment. The important assumption is that we need a solution for a scenario where a single user may have access to heterogeneous resources, with different access methods, possibly distributed across multiple Grid infrastructures which run different middleware and belong to separate virtual organizations. In this section we demonstrate how H2O middleware may be used for creating a user's virtual resource pool, which can then be used for deployment and executing the application on the created infrastructure without the need for systems administrator involvement. In the following subsections we describe in more detail our approach to:

- setting up the user's virtual resource pool,
- executing the application on the created infrastructure.

A. Infrastructure setup

Our approach to creating the inter-Grid, user-centric virtualization layer involves a unified fabric of (transient) H2O kernels spawned dynamically on Grid-enabled resources. For standalone machines this may be done directly, using SSH, whereas on a cluster or on a remote Grid resources there is a need to submit an H2O kernel as a batch computing job. When the kernel is started, the resource becomes a configurable element of the user's virtual pool (see Fig. 2). In the Grid infrastructure with an automatic resource broker the location of the computing node on the remote site where the kernel is running may be unknown to the user in advance so we proposed a simple discovery mechanism to locate available kernels. As a discovery service we have used the Java Naming and Directory Interface (JNDI)-enabled Harness Distributed Naming Service (HDNS) system [20] running on a centralized server, however it is also possible to run multiple HDNS servers to provide fault tolerance. In addition to the discovery service, we have elaborated a simple scheduling mechanism to automate the process of assigning components to available resources for deployment.

An important aspect of our approach is that we do not require any specific software pre-installation on the target computing nodes, which might involve huge administrative overhead, especially in the case of multiple Grid systems. Consequently, our approach allows seamless installation of the required base software on the computing machines. This solution stems from the nature of the distribution of H2O software, which is designed to run out-of-the-box on any Java-enabled system. We have prepared a preconfigured thin version of H2O distribution which also contains the MOCCA library and fits into a 4MB archive file. All that is required to install and run the H2O kernel, is to unpack the distribution and run the specified kernel executable. When accessing the target system using SSH or PBS on a local cluster the installation step can be done manually by the user. For a Grid infrastructure such as LCG [21], manual installation is not possible, because access to worker nodes of computing elements is available only through a job submission mechanism, i.e. via a Resource Broker, Globus and a local queueing system. Setting up the H2O kernel is then achieved by submitting a batch job which transfers this distribution on the working node, unpacks it and runs the kernel executable. The only requirement is that a Java virtual machine must be installed (JVM is present in most cases). We can note, that if the specified version of JVM is not available on the Grid site, it will be possible to add the Java Runtime Environment installation as a first step of the job. If the need for transferring large files becomes an issue, we can exploit the possibility of using the Grid replica management system to optimize file transfer time.

Another aspect we have to take into account is that when an automatic Grid Resource Broker decides for a Grid node to run a batch job, the location of the H2O kernel started by this job may not be known to the user in advance. Locating such a kernel requires some form of a discovery mechanism. For this purpose we use the HDNS naming service, which needs to be set up by the user prior to running H2O kernels. Subsequently, when the preconfigured H2O kernel is started, it automatically loads the naming service client pluglet. Its role is to contact the discovery service and to register the new kernel on the server. The client contacts the HDNS server using JNDI API, which gives the possibility of future integration with other naming services, e.g. ones based on the LDAP protocol.

It is worth mentioning that the first step does not mandate the usage of a component model for application programming. Furthermore, both steps may also be performed by different actors, so we can distinguish the resource pool provider from the application deployer or even from the final user, just like in H2O.

We should also point out the security aspects. In our current experiments we use an open system, which may potentially lead to unauthorized access to the pool of user's resources. However, it is possible to guarantee system security by means of an H2O kernel access policy. One of the option feasible for our scenario would be to preconfigure all kernels which belong to the user's pool in such a way that they accept only code signed by the specified user's digital certificate, consequently denying access to other parties. This would be in agreement with our user-centric approach, since it is the user who sets up the H2O kernels and decides who can access a given pool of resources.

In our current experiments, management of the pool of H2O kernels has to be performed manually by the user. We have written a set of scripts which may be used for submission of H2O kernels using the PBS system and Job Description Language (JDL) scripts for LCG middleware. A set of our simple command-line tools may be used for querying the Name Service, testing the connectivity to remote kernels and performing shutdown of the pool. For monitoring of specific kernels, the H2O GUI can be used, yielding information on the state of deployed components. As our experiments with setting up a pool of H2O kernels have given promising results, we plan to develop more user-friendly tools which will automate the process of management.

B. Application deployment and execution

Once the required number of H2O kernels is running and registered in the discovery service, application execution may take place. It is worth noticing that when using the Grid infrastructure, whenever the pool of kernels is created, it will most probably contain different machines, chosen by the resource broker. In order to run the application on such a testbed, the user should prepare the components and the application description in the following way:

- each component of the application should be available as a JAR file published on the HTTP server,
- the script which creates the component instances connects them and finally triggers application execution.

Since the list of available H2O kernels is not known before runtime, there is a need to use some form of a *scheduler* be responsible for automatic selection of locations for running the component instances. The scheduler contacts the discovery service to query about the available H2O kernels and selects them according to a specific policy. Whereas various approaches to component deployment are being researched [16], for the purpose of our experiments we have implemented a simple scheduler prototype which selects the kernels based on a *round-robin* policy. The scheduler may be invoked by the deployment script, and the returned locations of kernels can be then passed to the *BuilderService* for creating single or multiple instances of components.

Once the components are instantiated on remote sites and their ports are connected, the script invokes the starter component and passes control to the application components. It is also worth mentioning that in order to receive application results, possibly even in an interactive way, it is preferable to run one of the components on the kernel, locally available to the user. This provides online access to application logs and output, whereas inorder to obtain access to the files produced on Grid worker nodes, it is necessary to use additional Grid file transfer tools.

V. Case study - simulation of gold clusters formation

The feasibility of the approach presented above was demonstrated with an application that simulates formation of gold atom clusters which is a very important phenomenon in the field of nanoscale devices. Modeling of clusters involves several energy minimization methods which are highly compute-intensive. The computing resources used for these experiments comprised a computer cluster with a PBS batch system at the Academic Computer Centre CYFRONET-AGH and the CrossGrid testbed running LCG middleware [21].

A. Description of the application

Clusters of atoms are an interesting form in between isolated atoms or molecules and solid state; research in this field may therefore be very important for the technology of constructing nanoscale devices. Modeling of clusters involves several energy minimization methods such as Molecular Dynamics Simulated Annealing (MDSA) or the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) method, as well as choosing an empirical potential [22]. These methods are highly compute-intensive, and an optimal result depends on the number of possible iterations and initial configurations for each simulation run.

The original application (sequential code written in C) has been rewritten in Java and its functional modules have been divided into separate components as presented in [23]. Componentization allows for more flexibility of application assembly when experimenting with different minimization methods, and the performance penalty incurred by using Java

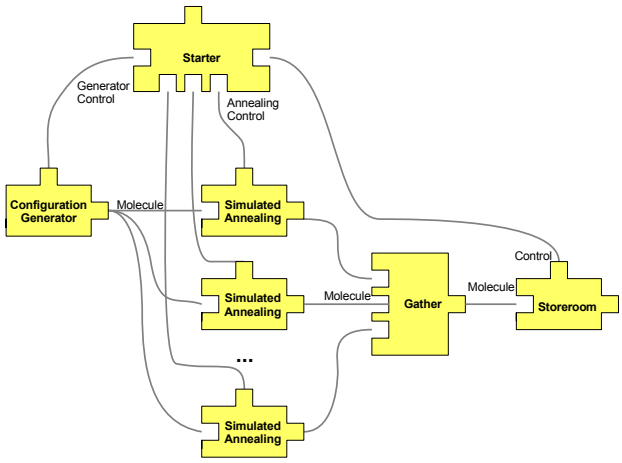

Fig. 3. Configuration of components in the test application.

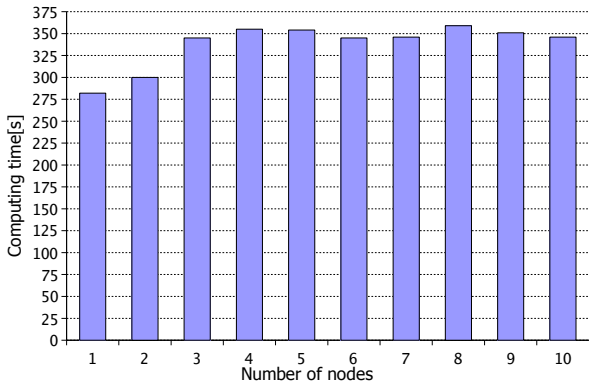

Fig. 4. Results achieved on the sample pool of heterogeneous resources, where the problem size grows with the number of computing nodes.

may be outweighed by the possibility of using distributed resources for computation, offered by the component framework, i.e. by MOCCA. Our previous results indicate the acceptable scalability of this application when tested on a single cluster [23].

For our experiments we used the component configuration as shown in Fig. 3. The *Starter* component is responsible for coordinating the work of other components. *Configuration Generator* creates the initial configurations of atoms, which are then consumed by multiple *Simulated Annealing* components, performing the actual minimization process. The *Configuration Generator* and *Simulated Annealing* components may be used for both sequential and distributed configurations, since they do not have multiple ports. The *Storeroom* component is responsible for storing all achieved configurations and may be used to derive results statistics. A single *Molecule* port is defined to exchange data between components. The *Storeroom* component was initially designed to support a single *Molecule* provider, and to keep it simple, we have decided to add a separate *Gather* component which handles multiple connected components and passes their results to the *Storeroom*.

B. Experiments on the Grid

For experiments with a heterogeneous infrastructure setup we used two stand-alone machines at CYFRONET, Krakow, accessible directly via SSH a cluster node at CYFRONET, accessible using PBS, and the European CrossGrid testbed running LCG middleware with a Resource Broker located at LIP in Lisbon, Portugal. When submitting jobs via the Resource Broker we had to restrict the list of resources to only those Grid sites which allowed jobs (in our case – H2O kernels) to open TCP ports for incoming connections from the outside world. Therefore we were able to use cluster nodes at PSNC, Poznan and IFCA, Santander, Spain. The machines at CYFRONET were Intel Xeon CPU 2.40GHz computers, with 512 MB RAM running Red hat Linux 7.3, and similar configurations were available on the remote Grid sites. Sun J2SDK 1.4.1 was available on all machines, therefore no additional JVM installation was required.

For setting up the pool of H2O kernels we set up an HDNS server on one of the machines at CYFRONET, and on another machine we started the kernel on which the *Starter* component had to be deployed in order to have direct access to results and timing statistics for the application run. Subsequently we were able to spawn more H2O kernels on local cluster nodes and on Grid sites using scripts, as described in the previous section. Using the pool of H2O kernels we were able to deploy our distributed component application, creating one compute-intensive *Simulated Annealing* component for each available kernel. It should be noted note that once the pool of kernels was created, deploying the application was as simple as on a single cluster, requiring only minor changes to the deployment script. These changes involved replacing a static list of machines with an appropriate invocation of our scheduler functions.

Fig. 4 shows the sample application runtime depending on the number of computing nodes. On each of the nodes there was one *Simulated Annealing* component deployed. The total number of initial configurations for simulation was equal to the number of computing nodes, so each of the components was computing one full simulation run for a single molecule. By increasing the number of resources in the pool the user was able to simulate more molecules, which is important from the application's point of view (in order to gather better statistics). We can observe that the total computing time does not increase substantially with the number of nodes, which approximates an ideal case where the time should remain constant. We should note, however, that the goal of our tests was to show proof of concept for our approach to building a user-centric pool of resources for distributed component applications, and not to run systematic performance measurements on the Grid, where speedup is not as clear as on a parallel machine or on a single

cluster [24]. However the results achieved on this sample pool of resource, are quite promising for such an ad-hoc testbed and base upon the nature of the tested application (which was not communication-intensive).

VI. Conclusions and future work

In this paper we are dealing with a specific scenario, where users are willing to aggregate their computing resources accessible by means of accounts on single machines, accounts on clusters with batch scheduling systems or membership in Virtual Organizations for Grid infrastructures. These resources are needed to perform a single distributed application (possibly a compute-intensive one) which may be expressed as custom code, and, even more importantly, doesn't have to be pre-installed on the available resources.

We demonstrate that the selected programming model and the method of resource aggregation provide reasonable solutions for such a scenario. We show that the component model may be a good choice in this situation, due to its naturally-provided deployment mechanisms and the possibility of easy implementation of parallel computing constructs. Our experience with the CCA suggests that this model, while simple, is rich enough to allow building usable distributed applications and may be used for potentially more dynamic large-scale distributed computing scenarios.

In our experiments with H2O and MOCCA presented in this paper we show that a single user may create a pool of resources spanning multiple Grid and non-Grid machines, and successfully deploy and run component application on such a virtual infrastructure.

It is important to note that we were able to exploit the "legacy" CrossGrid testbed, without the need for using system administrators privileges and at the acceptable cost of the effort of one user responsible for manual pool setup and management. This is possible thanks to the H2O kernel being a lightweight container, suitable for creating a higher virtualization layer on top of a batch-oriented Grid infrastructure. This "legacy" infrastructure provides, in return, mechanisms for automatic resource allocation and scheduling, as configured within the resource brokers and batch schedulers on involved sites.

During our experiments we also observed some limitations of CCA and our implementation, such as relying only on synchronous RMI and the lack of parallel constructs or ADL support explicitly in the model. We believe that the adoption of the GCM specification [13] elaborated within the CoreGRID project could help overcome these issues and open an interesting research area involving component model interoperability.

We are also aware of the fact that the process of setting up the pool of resources which we have proposed could potentially be fully automated, so that the user would not need to know the details of the underlying infrastructure – merely provide a pointer to the resources and credentials required to access them. This is also one of directions for future development, together with elaboration of support for more dynamic changes of the pool at application runtime.

Another issue which remains to be solved involves the network connectivity problem which we encountered when trying to access computing nodes belonging to private networks or hidden behind firewalls. We believe that this may be solved via our experience with the JXTA peer-to-peer framework [25], which can be used to create an overlay network spanning computing nodes from private sub-networks of Grid clusters, as well as idle desktop machines.

Acknowledgements: This work was partially supported by the EU IST CoreGRID Project. The authors are grateful to Piotr Nowakowski for his remarks.

References

[1] R. Armstrong, G. Kumfert, L. C. McInnes, S. Parker, B. Allan, M. Sottile, T. Epperly, and T. Dahlgren, "The cca component model for high-performance scientific computing," *Concurr. Comput. : Pract. Exper.*, vol. 18, no. 2, pp. 215–229, 2006.

[2] M. Malawski, D. Kurzyniec, and V. Sunderam, "MOCCA – towards a distributed CCA framework for metacomputing," in *Proceedings of the 10th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS2005) in conjuncion with International Parallel and Distributed Processing Symposium (IPDPS), Denver, USA, April 2005*. IEEE, 2005.

[3] D. Kurzyniec *et al.*, "Towards Self-Organizing Distributed Computing Frameworks: The H2O Approach," *Parallel Processing Lett.*, vol. 13, no. 2, pp. 273–290, 2003.

[4] M. Humphrey, G. Wasson, K. Jackson, J. Boverhof, M. Rodriguez, J. Bester, J. Gawor, S. Lang, I. Foster, S. Meder, S. Pickles, , and M. McKeown, "State and events for web services: A comparison of five WS-resource framework and WS-notification implementations," in *4th IEEE International Symposium on High Performance Distributed Computing (HPDC-14)*, Research Triangle Park, NC, 24-27 July 2005.

[5] V. Getov and T. Kielmann, Eds., *Component Models and Systems for Grid Applications.* Springer, 2005.

[6] F. Bertrand, R. Bramley, A. Sussman, D. E. Bernholdt, J. A. Kohl, J. W. Larson, and K. B. Damevski, "Data redistribution and remote method invocation in parallel component architectures," in *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers*, 2005, p. 40b.

[7] C. Perez, T. Priol, and A. Ribes, "A parallel corba component model for numerical code coupling," *The International Journal of High Performance Computing Applications (IJHPCA)*, vol. 17, no. 4, pp. 417–429, 2003, special issue Best Applications Papers from the 3rd Intl. Workshop on Grid Computing.

[8] D. Gannon, R. Bramley, G. Fox, S. Smallen, A. Rossi, R. Ananthakrishnan, F. Bertrand, K. Chiu, M. Farrellee, M. Govindaraju, S. Krishnan, L. Ramakrishnan, Y. Simmhan, A. Slominski, Y. Ma, C. Olariu, and N. Rey-Cenvaz, "Programming the Grid: Distributed software components, P2P and Grid web services for scientific applications," *Cluster Computing*, vol. 5, no. 3, pp. 325 – 336, Jul 2002.

[9] R. Olejnik, B. Toursel, M. Tudruj, and E. Laskowski, "Optimized java computing as an application for desktop grid," in *Proceedings of the 4th Cracow Grid Workshop*. Krakow, Poland: ACC CYFRONET-AGH, 2005.

[10] F. Baude *et al.*, "From distributed objects to hierarchical grid components," in *Int. Symp. on Distributed Objects and Applications (DOA), Catania, Italy*, ser. LNCS, vol. 2888. Springer, 2003, pp. 1226 – 1242.

[11] M. Aldinucci, S. Campa, M. Coppola, M. Danelutto, D. Laforenza, D. Puppin, L. Scarponi, M. Vanneschi, and C. Zoccolo, "Components for High Performance Grid Programming in Grid.IT," in *Proceedings of the Workshop on Component Models and Systems for Grid Applications, ICS '04 Intl. Conf.*, ser. CoreGRID, V. Getov and T. Kielmann, Eds. Springer, 2005, pp. 19–38.

[12] J. Duennweber and S. Gorlatch, "HOC-SA: A grid service architecture for higher-order components," in *Services Computing, 2004 IEEE Int. Conf. on (SCC'04)*. Shanghai, China: IEEE, 2004, pp. 288–294.

[13] "Proposals for a grid component model," Tech. Rep., 2004. [Online]. Available: http://www.coregrid.net

[14] S. Lacour *et al.*, "Deploying CORBA components on a computational grid," in *Component Deployment: 2nd Int. Working Conf., CD 2004, Edinburgh, UK, Proc.*, ser. LNCS, W. Emmerich *et al.*, Eds., vol. 3083. Springer, 2004, pp. 35 – 49.

[15] F. Baude, D. Caromel, L. Mestre, F. Huet, and J. Vayssière, "Interactive and descriptor-based deployment of object-oriented grid applications," in *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*. Edinburgh, Scotland: IEEE Computer Society, July 2002, pp. 93–102.

[16] M. Coppola, M. Danelutto, S. Lacour, C. Perez, T. Priol, N. Tonellotto, and C. Zoccolo, "Towards a common deployment model for grid systems," in *CoreGRID Workshop on Integrated research in Grid Computing*, S. Gorlatch and M. Danelutto, Eds. Pisa, Italy: CoreGRID, IST, November 2005, pp. 31–40.

[17] P. Graham, M. Heikkurinen, J. Nabrzyski, A. Oleksiak, , M. Parsons, H. Stockinger, K. Stockinger, M. Stroinski, and J. Weglarz, "EU Funded Grid Development in Europe," in *Grid Computing: Second European AcrossGrids Conference, AxGrids 2004, Nicosia, Cyprus, January 28-30, 2004. Revised Papers*, ser. Lecture Notes in Computer Science, M. D. Dikaiakos, Ed., vol. 3165. Springer, Jan. 2004, pp. 1–10.

[18] "GLUE information model," 2005. [Online]. Available: http://infnforge.cnaf.infn.it/glueinfomodel/

[19] UniGrids Project, "Uniform interface to grid services." [Online]. Available: http://www.unigrids.org/

[20] D. Gorissen, P. Wendykier, D. Kurzyniec, and V. Sunderam, "Integrating grid information services using JNDI," in *6th IEEE/ACM International Workshop on Grid Computing (Grid 2005)*, Seattle, Washington, USA, Nov. 2005.

[21] J. Gomes, M. David, J. Martins, L. Bernardo, A. García, M. Hardt, H. Kornmayer, J. Marco, R. Marco, D. Rodríguez, I. Diaz, D. Cano, J. Salt, S. Gonzalez, J. Sánchez, F. Fassi, V. Lara, P. Nyczyk, P. Lason, A. Ozieblo, P. Wolniewicz, M. Bluj, K. Nawrocki, A. Padee, W. Wislicki, C. Fernández, J. Fontán, Y. Cotronis, E. Floros, G. Tsouloupas, W. Xing, M. D. Dikaiakos, J. Astalos, B. A. Coghlan, E. Heymann, M. A. Senar, C. Kanellopoulos, A. Ramos, and D. Groen, "Experience with the international testbed in the crossgrid project." in *Advances in Grid Computing - EGC 2005, European Grid Conference, Amsterdam, The Netherlands, February 14-16, 2005, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 3470. Springer, 2005, pp. 98–110.

[22] N. Wilson and R. Johnston, "Modelling gold clusters with an empirical many-body potential," *Eur. Phys. J. D*, vol. 12, pp. 161–169, 2000.

[23] M. Bubak, M. Malawski, and M. Placek, "Component-based Approach to Modeling Gold Clusters using MOCCA," in *Proceedings of the 5th Cracow Grid Workshop*. Krakow, Poland: ACC CYFRONET-AGH, 2006, to appear.

[24] A. Hoekstra and P. Sloot, "Introducing grid speedup gamma: A scalability metric for parallel applications on the grid," in *Advances in Grid Computing - EGC 2005*, ser. Lecture Notes in Computer Science, P. Sloot, A. Hoekstra, T. Priol, A. Reinefeld, and M. Bubak, Eds., vol. 3470. Springer, February 2005, pp. 245–249.

[25] P. Jurczyk, M. Golenia, M. Malawski, D. Kurzyniec, M. Bubak, and V. S. Sunderam, "Enabling Remote Method Invocations in Peer-to-Peer Environments: RMIX over JXTA," in *Proceedings of PPAM 2005 International Conference*, ser. LNCS, R. Wyrzykowski, Ed., vol. 3911. Poznan: Springer, Sep 2005, to appear.