Adaptive Application Composition in Quantum Chemistry

Li Li[1], Joseph P. Kenny[2], Meng-Shiou Wu[3], Kevin Huck[4], Alexander Gaenko[3], Mark S. Gordon[3], Curtis L. Janssen[2], Lois Curfman McInnes[1], Hirotoshi Mori[5], Heather M. Netzloff[3], Boyana Norris[1], and Theresa L. Windus[3]

[1] Argonne National Laboratory, Argonne, IL
[2] Sandia National Laboratories, Livermore, CA
[3] Ames Laboratory, Ames, IA
[4] University of Oregon, Eugene, OR
[5] Ochanomizu University, Japan

Abstract. Component interfaces, as advanced by the Common Component Architecture (CCA), enable easy access to complex software packages for high-performance scientific computing. A recent focus has been incorporating support for computational quality of service (CQoS), or the automatic composition, substitution, and dynamic reconfiguration of component applications. Several leading quantum chemistry packages have achieved interoperability by adopting CCA components. Running these computations on diverse computing platforms requires selection among many algorithmic and hardware configuration parameters; typical educated guesses or trial and error can result in unexpectedly low performance. Motivated by the need for faster runtimes and increased productivity for chemists, we present a flexible CQoS approach for quantum chemistry that uses a generic CQoS database component to create a training database with timing results and metadata for a range of calculations. The database then interacts with a chemistry CQoS component and other infrastructure to facilitate adaptive application composition for new calculations.

1 Introduction

As computational science progresses toward ever more realistic multiphysics applications, no single research group can effectively select or tune all components of a given application, and no solution strategy can seamlessly span the entire spectrum of configurations efficiently. Common component interfaces, along with programming language interoperability and dynamic composability, are key features of component technology that enable easy access to suites of independently developed algorithms and implementations. By means of the Common Component Architecture (CCA) [1,2], such capabilities are now making inroads in scientific computing. The challenge then becomes how to make the best choices for reliability, accuracy, and performance, both when initially composing and configuring a component application, and when dynamically adapting to respond to continuous changes in component requirements and execution environments.

Computational quantum chemistry is a mature domain of scientific computing populated by numerous software packages offering a range of theoretical methods with a variety of implementation approaches. These packages provide a vast array of tools to be employed by practitioners who are often not developers or knowledgeable about the implementation details of these packages. The existence of certain methods as well as their performance on various types of hardware varies greatly within these packages, and the optimal configuration of these calculations is often a matter of trial and error, at least until a great deal of experience is accumulated with each package. For example, as the number of cores increases on commodity processors, memory bandwidth limitations will likely limit the number of cores that can be used effectively per socket to significantly fewer than the number available. Furthermore, predicting runtime can be useful for planning and queue management when running unfamiliar job types.

The challenges of efficiently employing and configuring quantum chemistry packages, faced also in combustion, fusion, and accelerator modeling [3], motivate the design and implementation of generic support for *computational quality of service* (CQoS) [4], or the automatic composition, substitution, and dynamic reconfiguration of components to suit a particular computational purpose and environment. CQoS embodies the familiar concept of quality of service in networking as well as the ability to specify and manage characteristics of the application in a way that adapts to the changing (computational) environment. CQoS expands on traditional QoS ideas by considering application-specific metrics, or metadata, which enable the annotation and characterization of component performance. Before automating the selection of component instances, however, one must be able to collect and analyze performance information and related metadata. The two main facets of CQoS tools, therefore, are measurement and analysis infrastructure and control infrastructure for dynamic component replacement and domain-specific decision making. This paper focuses on the performance and metadata management and analysis support in CQoS infrastructure.

We present in this paper recent work by members of the CCA Forum and the Quantum Chemistry Science Application Partnership (QCSAP) [5], which includes developers of several leading high-performance quantum chemistry codes (GAMESS [6], MPQC [7], and NWChem [8]), to utilize the new CQoS infrastructure to guide adaptive runtime composition and performance optimization of component-based parallel quantum chemistry applications. Parallel application configuration has driven the initial development and integration of CQoS infrastructure in the QCSAP, laying the groundwork for more sophisticated analysis to configure algorithmic parameters for particular molecular targets, calculation approaches, and hardware environments.

The remainder of this paper is organized as follows. Section 2 provides an overview of related work; Section 3 highlights key features of the Common Component Architecture and its use in high-performance quantum chemistry. Section 4 introduces our CQoS approach for quantum chemistry, and Section 5 reports on some preliminary experiments. Section 6 provides concluding remarks and discussion of future work.

2 Related Work

Adaptive software architecture is an area of emerging research, as evidenced by numerous recent projects and related work [9–21]. Many approaches to addressing different aspects of adaptive execution are represented in these projects, from compiler-based techniques to performance model-based engineering approaches and development of new adaptive numerical algorithms.

Unlike these efforts, our approach specifically targets large-scale parallel computations and support of interoperability among scientific packages. In designing our CQoS database interfaces and middleware components, we rely on the existing high-performance infrastructure provided by the CCA, in which multiple component implementations conforming to the same external interface standard are interoperable, and the runtime system ensures that the overhead of component substitution is negligible.

A large number of tools for performance analysis exist, including TAU [22], Prophesy [23], and SvPablo [24]. Each tool defines its own performance data representation and storage, from custom ASCII representations or XML to SQL, DB2, or Oracle databases. Efforts are under way to define common data representations for performance data; however, we expect that the standardization will take some time and it will be longer before tools implement mappings from their native formats to the standard one. Thus, we have focused on defining *interfaces* for querying and manipulating the data, which can then be implemented as components mapping to different representations. To our knowledge, the research discussed in this paper is the first attempt to provide language-independent component interfaces and corresponding implementations for performance database manipulation, specifically targeting parallel scientific applications. This approach supports multiple underlying representations and does not preclude the use of non-component performance analysis tools.

A rich set of performance tools [25–27], including PerfExplorer [28], aim to improve the execution behavior of a program based on information on its current or previous runtime behavior. The tools, however, use low-level compiler-based techniques or are restricted to specific parallel computer systems or application domains. In contrast, we integrate PerfExplorer into the CCA infrastructure to support adaptation in generic parallel scientific applications.

3 Common Component Architecture and Quantum Chemistry

CCA Overview. This work leverages the component standard for scientific computing under development by the CCA Forum. Component technology (see, e.g., [29]), which is now widely used in mainstream computing but has only recently begun to make inroads in high-performance computing (HPC), extends the benefits of object-oriented design by providing coding methodologies and supporting infrastructure to improve software's extensibility, maintainability, and reliability.

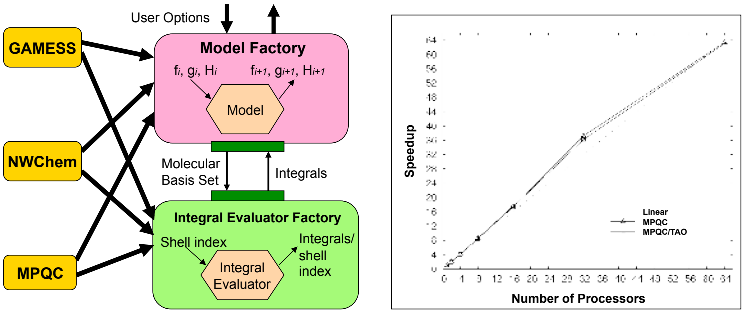

Fig. 1. *Left:* CCA component approach for a quantum chemistry application. *Right:* Isoprene HF/6-311G(2df,2pd) parallel speedup in MPQC-based CCA simulations of molecular geometry; the plot shows nearly linear speedup on 1 through 64 processors both when using MPQC alone and when using external numerical optimization components in TAO.

The CCA Forum is addressing productivity challenges of diverse scientific research teams by developing tools for plug-and-play composition of applications in parallel and distributed computing. The core of this work is a component model and reference implementation [2] tailored to the needs of high-end scientific computing. Key features are language-neutral specification of common component interfaces, interoperability for software written in programming languages important to scientific computing, and dynamic composability, all with minimal runtime overhead.

The specification of the Common Component Architecture defines the rights, responsibilities, and relationships among the various elements of the model. Briefly, the elements of the CCA model are as follows:

- *Components* are units of software functionality that can be composed together to form applications. Components encapsulate much of the complexity of the software inside a black box and expose only well-defined interfaces.
- *Ports* are the interfaces through which components interact. Components may provide ports, meaning that they implement the functionality expressed in a port (called *provides ports*), or they may use ports, meaning that they make calls on a port provided by another component (called *uses ports*).
- *Frameworks* manage CCA components as they are assembled into applications and executed. The framework is responsible for connecting *uses* and *provides* ports.

Quantum Chemistry Components. The QCSAP has adopted a component architecture based on the CCA. Both coarse-grain componentization (where a component encompasses a task such as energy, gradient, or Hessian evaluation) and fine-grain componentization (where a component computes only integrals) are incorporated [30, 31]. The left-hand side of Figure 1 illustrates how common component interfaces for molecules, models, basis sets, and integral evaluation facilitate the sharing of code among these three chemistry teams.

The component approach also enables the chemistry teams to leverage external capabilities in the wider scientific community. For example, as shown in the right-hand side of Figure 1, chemists have achieved scalable performance in MPQC-based CCA simulations for molecular shape determination using parallel components from the Toolkit for Advanced Optimization (TAO) [30].

4 CQoS Infrastructure and Its Application in High-Performance Quantum Chemistry

Scientific computations often require the specification of many options and parameters. Quantum chemical options, for instance, include the basic selection of methods, expansion basis, and convergence criteria, as well as low level details such as hardware configuration and algorithmic parameters. The initial focus of CQoS tools is parallel application configuration to effectively exploit high-performance architectures. The difficulty of configuring parallel computations is compounded by the proliferation of multicore processors, resulting in three levels of processing elements (nodes, processors/sockets, and processor cores), and the variety of hardware environments, ranging from networks of workstations for development to massively parallel machines for production runs.

A key aspect of CQoS research is generating and collecting meaningful data for storage in a database and subsequent analysis by performance tools. Full automation of the processes of collecting and managing performance data and metadata, building a performance model, and conducting detailed evaluation is beyond the scope of this paper. We focus on coarse-grain computations, where we select several major parameters that can affect the performance of scientific codes and then use the performance data to enable adaptive application configuration.

While motivated by adaptivity in different problems (quantum chemistry, combustion, fusion, and accelerator simulations), we believe that the *infrastructure* for analyzing and characterizing the problems and determining and invoking solution strategies will indeed be similar for such large-scale scientific simulations. We introduce nonfunctional properties and application-specific information, or metadata, into performance analysis and decision-making. Metadata include algorithm or application parameters, such as problem size and physical constants, compiler optimization options, and execution information, such as hardware and operating system information. Ideally, for each application execution, the metadata should provide enough information to be able to repeat the run. The metadata can help classify performance measurements and provide clues for tuning performance. CQoS infrastructure, therefore, includes database components that manage performance data and associated metadata, comparator components that support query and extraction of data, and analysis components that conduct performance analysis to suggest adaptation strategies. Figure 2 illustrates the utilization of the CQoS components (described in more detail in Sections 4.2 and 4.3) in an adaptive quantum chemistry application. Thanks to our uniform quantum chemistry interfaces, the capabilities for automated adaptive configuration shown in Figure 2 are available to all three

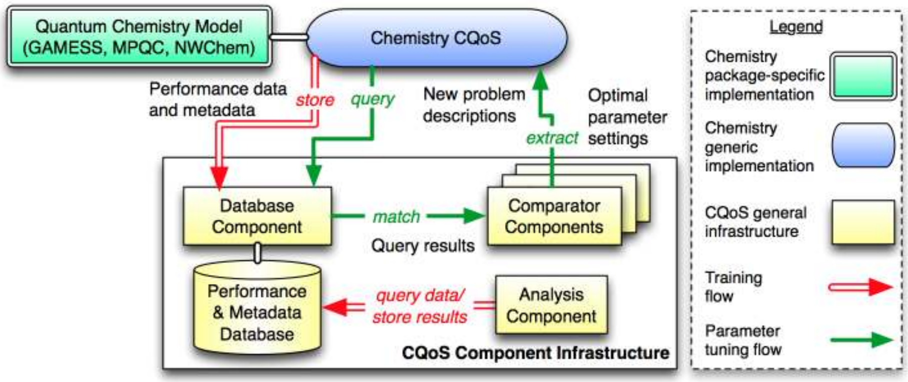
Quantum Chemistry Model (GAMESS, MPQC, NWChem)
Chemistry CQoS
Performance data and metadata
store
query
New problem descriptions
extract
Optimal parameter settings
Database Component
match
Query results
Comparator Components
Performance & Metadata Database
query data/ store results
Analysis Component
CQoS Component Infrastructure
Legend
Chemistry package-specific implementation
Chemistry generic implementation
CQoS general infrastructure
Training flow
Parameter tuning flow

Fig. 2. Components and parameter tuning flow in a CQoS-enabled quantum chemistry application.

QCSAP quantum chemistry codes (GAMESS, MPQC, and NWChem), as well as any other applications that adopt these interfaces in the future.

As shown in Figure 2, our CQoS approach to automating parallel application configuration involves three phases: collection of performance data in a training database, performance analysis, and adaptive application composition based on this information. The training data for quantum chemistry contains timing results for calculations spanning a range of molecular characteristics and hardware configurations. The intermediate chemistry CQoS component bridges domain-specific chemistry components and generic CQoS tools. It uses the general CQoS database component interfaces to store and query performance and associated metadata, which in this case consist of the application's parallel configuration parameters. Once this database is populated on a target machine, users can request an appropriate configuration for a new molecule and calculation type from the chemistry CQoS component. Furthermore, we have defined comparator components that serve as filters when searching for appropriate parameter settings in the database. An analysis component based on PerfExplorer [28] provides offline performance analysis to identify sources of parallel inefficiency and determine appropriate parameters for a given configuration. The remainder of this section discusses these phases in more detail.

4.1 Training Data Collection for Scientific Components

The granularity of a component can significantly affect the usefulness of the performance data collected. Without detailed data, such as time spent in I/O or communication, the component-level performance data from a coarse-grain componentized computation may not provide insights into why the component does not scale well under some circumstances. This situation may create difficulties in developing adaptive CQoS strategies.

Generating and collecting meaningful data for subsequent analysis by performance tools is not as straightforward as one might expect, however, and the task is especially challenging for high-performance scientific applications like quantum chemistry. Many parameters in quantum chemistry computations can affect the efficiency and accuracy of a computation; moreover, it is not always obvious how to quantify some of these parameters. The complexity of modern HPC architectures only exacerbates the difficulty of finding appropriate parameters to achieve the best results (or tradeoff between accuracy and efficiency).

In order to acquire more detailed performance data for CQoS research, some supporting tools have been developed that facilitate data collection and management [32]. In addition, CCA-compliant TAU-based performance monitoring components [33] can be employed to collect performance data for computational components.

4.2 Database Components and Their Usage in Data Training

The database component interface design is intended to support the management and analysis of performance and application metadata, so that the mapping of a problem to a solution that can potentially yield the best performance can be accomplished statically or at runtime. The UML diagram in Figure 3 shows the main interfaces and some of their methods.

We introduce two types of components for storing and querying CQoS performance data and metadata. The database component provides general-purpose interfaces for storing and accessing data in a physical database. The comparator interfaces compare and/or match properties of two problems under user-specified conditions.

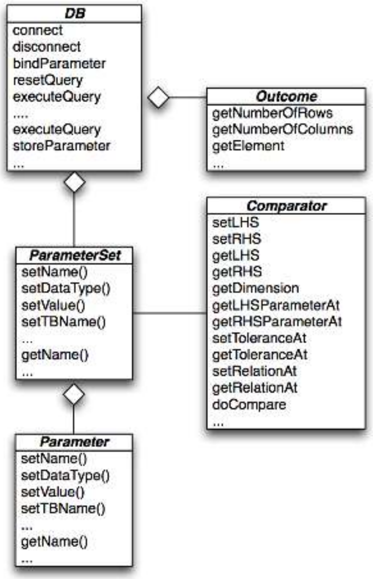

Fig. 3. UML diagram of CQoS database component interfaces and methods.

Comparator Components. Three sets of interfaces are associated with a comparator component: *Parameter*, *ParameterSet*, and *Comparator*. A *Parameter* captures a single property of a problem, for example, the count of a specific atom type in a molecule. A *Parameter*, which is described by its name, data type, and value, is associated with a table in the physical database. The *Parameter* interfaces also support comparisons against another peer parameter under user-specified conditions. A *ParameterSet* represents a group of related parameters, for example, a set of parameters that characterizes a molecule or a set of scalar or Boolean linear system properties. Using the *ParameterSet* interfaces, users can create and manage parameter set members. When selecting a solution method, time-dependent problem or system properties are described as one or more *ParameterSets*. The user or an automated adaptive heuristic can then match the formatted parameter sets to a database to determine the best solution method or configuration. A *Comparator* defines the rules to compare two sets of parameters. For instance, a *Comparator* can determine the closeness of two sets of parameters (i.e., whether they are within ϵ of each other).

Database Components. There are two classes of interfaces associated with a database component, *DB* and *Outcome*. The application connects to a database component by using the *DB* port, which handles (potentially remote) database connections, queries, and storage and retrieval of parameters and parameter sets. The *DB* interface also supports the query of experimental runs having parameter sets that satisfy user-specified conditions (e.g., limiting the parameter set to a range of values). The *Outcome* interface supports transformation of database results returned from a DB query to user-readable format, as well as access to the individual data elements.

During the training phase of the CQoS process for quantum chemistry, performance statistics and application metadata for selected problem instances are added into the database. This training data can then be used for future performance analysis and solution method matches, as further discussed in Section 4.3. Before the execution of an application, application-specific *Comparator* implementations help match the initial problem properties and system states against historical information to find a good initial solution method. During runtime, time-dependent application and system characteristics are captured in metadata parameter sets. At runtime the *Comparator* implementation can dynamically match the metadata against a lightweight runtime database to determine the best-known method corresponding to the current application state.

4.3 Performance Analysis

We incorporated PerfExplorer [28] into CQoS infrastructure to support performance analysis and decison-making for runtime adaptivity. PerfExplorer, a framework for parallel performance data mining and knowledge discovery in the TAU performance system, was developed to facilitate analysis on large collections of experimental performance data. The framework architecture enables the development and integration of data-mining operations that can be applied to parallel performance profiles. The data repository for PerfExplorer is PerfDMF [34], a performance data management framework that integrates and interfaces to many common database management systems, including the CQoS training database. PerfExplorer is designed to make the process of analyzing large numbers of parallel performance profiles manageable. Dimension reduction methods such as clustering and correlation allow meaningful analysis of large data sets. PerfExplorer does not directly implement these techniques; rather,

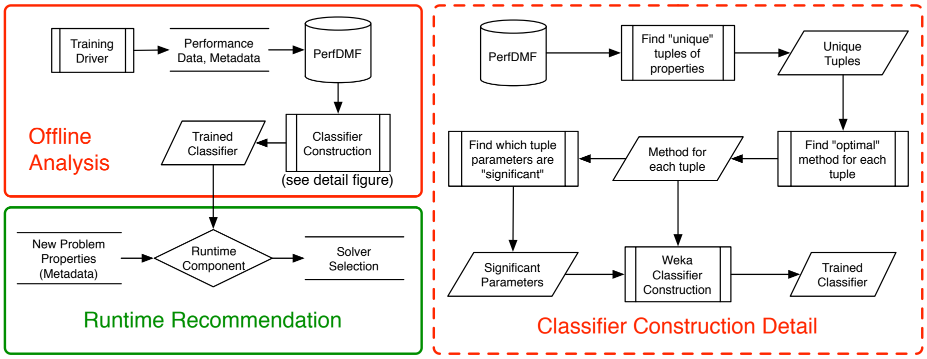

Fig. 4. PerfExplorer classifier construction.

it is integrated with existing analysis toolkits (e.g., Weka [35]) and provides for extensions using those toolkits. One such extension is to use classification capabilities in the Weka data-mining package to construct a runtime parameter recommendation system. Classification systems are a type of machine learning in which training data is input into a decision tree or space partitioning algorithm to construct a classifier. Classifiers belong a particular class of machine learning known as supervised learning, in which vetted training data with pre-selected attributes and known class types are used to train the classifier. In contrast, for exploratory, unsupervised learning methods such as clustering, class identifications are not known ahead of time. With a trained classifier, new test data can be identified as belonging to one of the identified classes. Classifiers can also be used to perform numerical prediction.

Figure 4 shows how a classifier for runtime recommendation is constructed and used. Training data for analysis is generated by executing the application multiple times, varying key parameters that have an effect on the total runtime. After the performance data and associated metadata for a set of training runs have been stored in the performance database, PerfExplorer loads the data and classifies it. A classifier is constructed using a simple Python script interface in which the application developer specifies independent application parameters and the dependent parameter, or class. Within the Python script, the classification method is also selected. Supported methods include alternating decision trees, support vector machines, and multilayer perceptrons (neural networks). Unique tuples of each combination of parameter values are found, and the best performing execution for each unique tuple is selected to represent that class. Optionally, Principle Components Analysis can be used to reduce the parameters to those that have the most influence over the variance in the data set. All classification is performed offline, as it can be a time intensive process. The results of the classification are stored in the form of a serialized Java object. This process can be performed either through PerfExplorer's GUI, the command line interface, or by using a CCA component wrapping PerfExplorer.

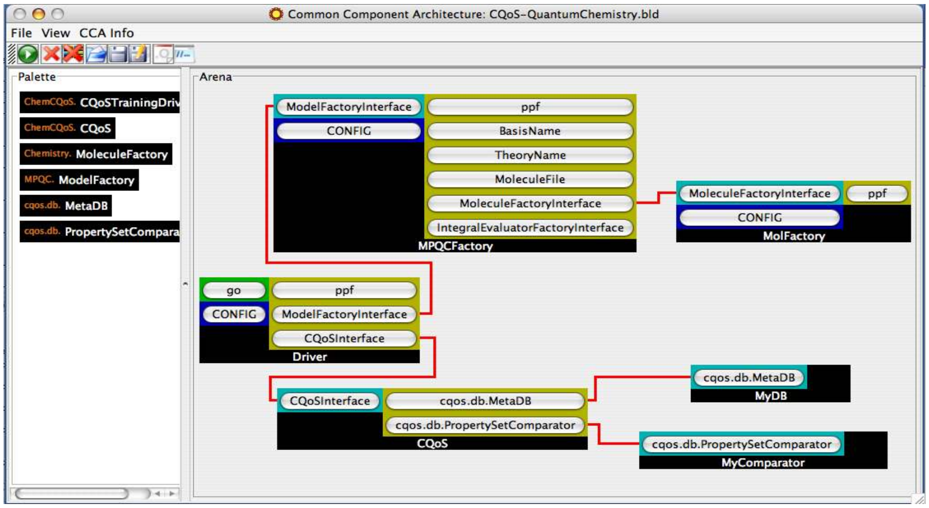

Fig. 5. Component wiring diagram for quantum chemistry showing usage of CQoS database components.

For production application runs, the classifier is loaded into a CCA component. The best parameter setting (class) is obtained by querying the classifier with the current values of the application-specific metadata. These values are matched to the classification properties to find the best class selection for the parameters. The runtime overhead of this step is minimal because it does not require access to the performance data database.

4.4 Adaptive Application Composition and Configuration

In application domains in which multiple software packages implement standard interfaces, we are able to capitalize on the CCA component approach and provide a single domain-specific CQoS component that manages interaction between generic CQoS infrastructure and various domain-specific implementations (in this case, GAMESS, MPQC, and NWChem), thereby reducing and simplifying the CQoS management code required in the domain-specific packages themselves. In Figure 5 a snapshot from the GUI of the Ccaffeine framework [36] illustrates the composition of a QCSAP quantum chemistry package (MPQC) and generic CQoS infrastructure components through a chemistry-specific CQoS component. The left-hand side of the figure shows the *palette* that contains available template components. In the wiring diagram on the right-hand side, a chemistry application instantiates both chemistry and CQoS components and connects so-called *uses* ports, as introduced in Section 3, and *provides* ports between related components.

The training driver component, *Driver*, manages chemistry model objects (e.g., *MPQCFactory* in Figure 5), acquires metadata from the model objects, and serializes interactions with the chemistry CQoS component, *CQoS*. During the training phase (see Section 4.2), the *Driver* populates a CCA type map, or dictionary, with metadata obtained from *MPQCFactory* describing algorithmic parameterization, hardware configuration, and application performance. For each training run, this metadata container is passed, along with a molecule descriptor class, to the chemistry CQoS interfaces, *CQoS*. The *CQoS* derives the metadata and handles the actual calls to the database component, *MyDB*, to store the metadata in the CQoS database.

For production runs, the *CQoS* passes metadata that describes a new calculation to the comparator component, *MyComparator*, to map the data to an application configuration that can potentially yield the best performance. Additionally, the CQoS infrastructure components can leverage the classification capabilities provided within the TAU package to obtain recommended configurations for calculations. The proposed production calculations are related to training calculations based on similarities in molecular properties and the resulting work load. Performance metrics associated with the training runs allow the TAU classifiers to recommend parameters, such as hardware configuration or algorithmic parameters, which aim to achieve properties such as good algorithmic performance (e.g., minimum iterations to solution), minimal time to solution, or maximum parallel efficiency. Once the TAU classifier has been constructed using training performance data and metadata, we do not need to build it again as the execution proceeds. Therefore the cost of training the classifier can be amortized by using it in many application runs.

The CQoS infrastructure can be used either for dynamic reconfiguration of an application during execution to respond to changing computing environments or for selection of initial parameters, with the goal of detemining how to maximize performance before the application runs. For runtime adaptation, the driver code checkpoints, collects the metadata describing the current computing state, and passes it to the TAU classifier. After the classifier suggests an optimal configuration, the driver can resume the execution by replacing with the recommended parameters. The period for checkpointing can be variable depending on the computing resource demand and program semantics. In an application where good initial parameter settings are essential for overall performance, we can use the CQoS infrastructure to determine appropriate initial parameter values based on the training or historical data. We expect that the predicted values will perform better than random values or at least as well as the values adopted by other researchers when performing experiments. We have partially evaluated this approach in experiments presented in the next section.

5 Experimental Results

As discussed in Section 1, computational chemistry applications rely on numerous software packages offering a range of theoretical methods and implementations. Our CQoS approach aims to automatically adapt these packages under these challenging situations, simplifying the tasks for end users.

5.1 MPQC

As an initial demonstration of the previously described CQoS architecture, the MPQC model was connected to the chemistry CQoS infrastructure and used to generate a small training data set. Hartree Fock energies were computed for five molecular structures obtained from the G2 neutral test set, which is commonly used for benchmarking. The five molecules selected were sulfur dioxide, disilane, vinyl chloride, acetic acid, and pyridine. For each molecule, the Hartree Fock energy was calculated using two selected basis sets, cc-pVTZ and aug-cc-pVQZ, with node counts ranging from 1 to 32 (as powers of 2) on the Catalyst cluster at Sandia Livermore (2-way Pentium 4 nodes with an Infiniband interconnection network). For these smaller calculations that cannot support high parallel efficiency at large scales, these node counts span from 100% parallel efficiency at low node counts to severe degradation of efficiency at larger node counts.

To demonstrate the potential for efficiency savings in this software environment, a Hartree Fock energy for the nitromethane molecule using the cc-pVTZ basis was chosen as a sample target calculation. We employed a very simple node count selection algorithm: selecting the training calculation with the nearest basis function count as the target calculation and then choosing the highest node count with parallel efficiency greater than 90% for that training calculation. Using this simplistic algorithm, the nitromethane target calculation achieves a parallel efficiency of 84%, which is 8% greater parallel efficiency than the next larger power of 2 node count. While these small sample calculations will not support efficient execution at large scales of parallelism, increasing the problem size only pushes these effects to larger node counts; the shapes of the efficiency curves and the potential efficiency gains using CQoS approaches will remain. With current environmental and energy efficiency concerns and yearly power budgets for modern HPC installations running into the millions of dollars, it seems clear that CQoS approaches should be a part of HPC efforts.

5.2 GAMESS

GAMESS, another application that computes Hartree Fock energies, has many options for solving the properties of the wavefunctions. There are two implementations for the energy solution, *conventional* and *direct*. The conventional implementation was written first but can be too resource intensive in terms of disk space and file I/O requirements on some systems. The direct version was developed to avoid storing intermediate integrals on disk, thus requiring some redundant computations, and in serial executions, is typically two to three times slower than the conventional. However, in parallel environments at higher processor counts, the direct method outperforms the conventional method due to excessive parallel and I/O overhead. The actual point where it makes sense to change methods depends on the wavefunction solution method, the input

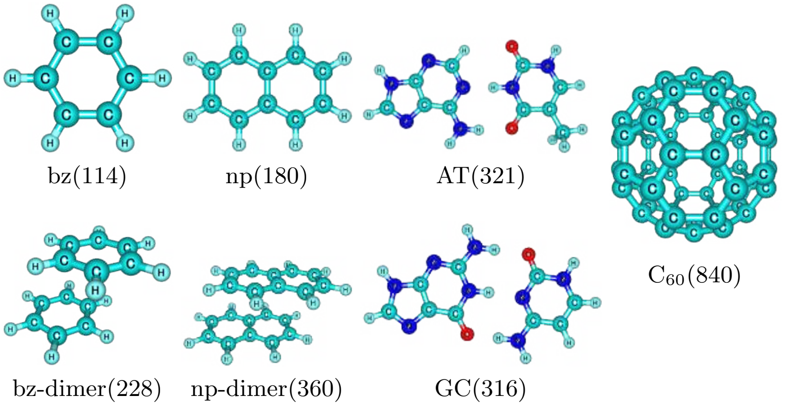

Fig. 6. Test cases: Benzene (bz) and its dimer, Naphthalene (np) and its dimer, Adenine-Thymine DNA base pair (AT), Guanine-Cytosine DNA base pair (GC), Buckminsterfullerene (C_{60}). In parentheses are the numbers of basis functions when using cc-pVDZ basis.

molecule or atom, the basis set, and the hardware. In addition, at one stage of the algorithm, a second-order Møller-Plesset correction (MP2 [37]) can be used to take into account the so-called "electron correlation." The MP2 method consumes more memory and disk space. With regard to the GAMESS application, one goal of the recommender system is to suggest whether to use the direct or conventional method, given the other parameter selections.

The initial set of molecules used for performance analysis is shown in Figure 6. Indicated numbers of basis functions (in parentheses) roughly correlate with resource demand of the corresponding computations: the greater the number of basis functions, the more demanding the computation is expected to be. The choice of molecules was based on their importance in chemistry and biology as well as on characteristic types of chemical interactions they represent; also, computations of molecules of a similar size (that is, with similar number of atoms and basis functions) are routine in contemporary quantum chemistry. The benzene and naphthalene molecules (labeled "bz" and "np" on the figure) represent fundamental aromatic systems. Their dimers (labeled "bz-dimer" and "np-dimer") represent models for π-π interactions believed to determine DNA stacking, protein folding, and other phenomena of great importance for biochemistry and chemistry. The pairs of DNA bases (labeled "AT" and "GC") are examples of hydrogen bonding; the interaction between the bases defines the double-helix structure of DNA. Finally, a molecule of buckminsterfullerene (labeled C_{60}) is taken as a representative of a large, highly symmetrical chemical structure, characteristic of carbon nanomaterials.

Hartree Fock energy was computed for each of these molecules, with and without MP2 correction, with various basis sets, and with varying numbers of nodes (up to 16) and processes per node (up to 8). The runs were computed on Bassi, an IBM p575 POWER5 system at the National Energy Research Scientific

Computing Center (NERSC). Bassi has 111 compute nodes with 8 processors and 32 GB of memory per node. This training data was used to construct a classifier in order to recommend whether to use the conventional or direct method to compute the energy. The independent parameters used to construct the classifier are shown in Table 1. There were 561 potential training instances, of which 150 of the best performing unique tuples were selected as the training set. The method (conventional or direct) used to generate the best performing tuple was used as the class for each training instance. A multilayer perceptron classifier was constructed using these instances.

Table 1. Parameters used for classifier construction.

Property	Training Values	Anthracene
# Cartesian Atomic Orbitals	120, 190, 240, 335, 340, 380, 470, 830	640
# Occupied Orbitals	21, 34, 42, 68	47
# Processes per Node	2, 4, 8	8
# Nodes	1, 2, 4, 8, 16	1,2,4,8,16
Second-order Møller-Plesset	disabled (MP0), enabled (MP2)	MP0, MP2

An eighth molecule, anthracene, was used to test the classifier. The test values for the parameters are also shown in Table 1. When used at runtime, the classifier recommended using the conventional method for the 1, 2, and 4 node runs (8, 16, and 32 processes, respectively), and using the direct method for the 8 and 16 node runs (64 and 128 processes). The empirical results from anthracene are shown in Figure 7. The classifier was correct in classifying 9 out of 10 instances − the 4 node direct MP2 run outperformed the conventional MP2 run, but only barely (300 seconds compared to 306 seconds). In all of the other configurations, the recommender correctly identified the method to use in order to achieve the fastest time to completion.

6 Conclusion and Future Work

This paper introduced a CQoS approach for quantum chemistry that leverages the CCA component environment to address new challenges being faced by applications teams when dynamically composing and configuring codes in high-performance computing environments. We have built prototype database components for managing performance data and associated metadata for high-performance component applications. These components are part of a larger CQoS infrastructure, which has the goal of enabling automated component selection and configuration of component-based scientific codes to respond to continuous changes in component requirements and their execution environments. We integrated performance analysis capabilities of PerfExplorer into the general CQoS infrastructure to classify performance and meta-information and then suggested appropriate configurations for new problem instances. The usage of

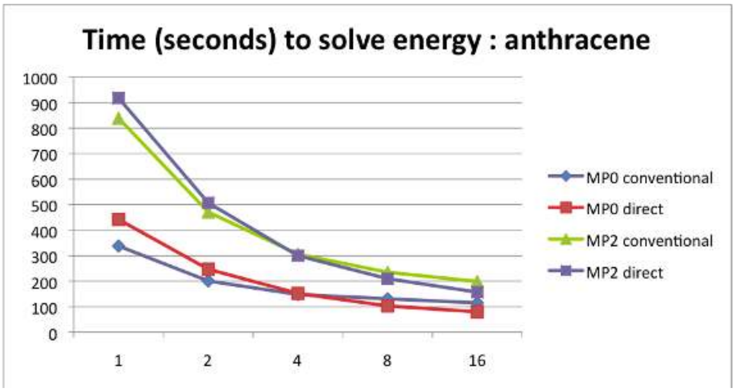

Fig. 7. Anthracene empirical results: Number of nodes vs. wall-clock time. Times are in seconds; less time indicates faster performance.

the CQoS infrastructure components in quantum chemistry applications demonstrates our initial success in adaptive parallel application configuration.

Our next steps in the quantum chemistry-specific part of this research include predicting, in addition to the runtime, the accuracy of the computations (with respect to energy or other properties). We will employ other classes of widely-used quantum chemical methods, starting with density functional theory and coupled clusters approaches. We will also incorporate new metadata fields whose values will be collected along with performance data, for example, parameters representing molecular symmetry. Furthermore, we plan to venture into the (rather unexplored) area of quantification of similarity between molecules. More generally, future work includes enhancing the CQoS infrastructure to support sophisticated analysis for (re)configuring algorithmic parameters and component instances during runtime, developing application-specific performance models, and incorporating the training CQoS phase into empirical experiment design.

We are also employing the new CQoS infrastructure to facilitate dynamic adaptivity of long-running simulations in other application domains, including parallel mesh partitioning in combustion and efficient solution of large-scale linear systems in fusion and accelerator models [3]. Our long-term goals are to define a comprehensive architecture for enabling CQoS in scientific simulations, consisting of general-purpose performance monitoring, analysis, and database middleware components, which can then be combined with easy-to-write domain-specific components for defining quality metrics and adaptation strategies.

Acknowledgments

We thank the reviewers of this paper for valuable feedback that helped us to clarify the presentation.

The CCA has been under development since 1998 by the CCA Forum and represents the contributions of many people, all of whom we gratefully acknowledge.

This work was supported in part by the Office of Advanced Scientific Computing Research via the Scientific Discovery through Advanced Computing (SciDAC) initiative [38], Office of Science, U.S. Department of Energy, under Contracts DE-AC02-06CH11357, DE-AC04-94AL85000, DE-AC02-07CH11358, and DE-FG02-07ER25826. This work is a collaboration among the Quantum Chemistry Scientific Application Partnership (QCSAP) [5], the Center for Component Technology for Terascale Simulation Software [39], and the Performance Engineering Research Institute [40].

References

1. Armstrong, R. et al.: Common Component Architecture (CCA) Forum. `http://www.cca-forum.org/`
2. Allan, B.A., Armstrong, R., Bernholdt, D.E., , Bertrand, F., Chiu, K., Dahlgren, T.L., Damevski, K., Elwasif, W.R., Epperly, T.G.W., Govindaraju, M., Katz, D.S., Kohl, J.A., Krishnan, M., Kumfert, G., Larson, J.W., Lefantzi, S., Lewis, M.J., Malony, A.D., McInnes, L.C., Nieplocha, J., Norris, B., Parker, S.G., Ray, J., Shende, S., Windus, T.L., Zhou, S.: A component architecture for high-performance scientific computing. Intl. J. High-Perf. Computing Appl. **20**(2) (2006) 163–202
3. McInnes, L.C., Ray, J., Armstrong, R., Dahlgren, T.L., Malony, A., Norris, B., Shende, S., Kenny, J.P., Steensland, J.: Computational quality of service for scientific CCA applications: Composition, substitution, and reconfiguration. Technical Report ANL/MCS-P1326-0206, Argonne National Laboratory (Feb. 2006)
4. Norris, B., Ray, J., Armstrong, R., McInnes, L.C., Bernholdt, D.E., Elwasif, W.R., Malony, A.D., Shende, S.: Computational quality of service for scientific components. In: Proc. Int. Symp. on Component-Based Software Engineering, Edinburgh, Scotland (2004)
5. Gordon, M. (PI): Chemistry Framework using the CCA. `http://www.scidac.gov/matchem/better.html`
6. Schmidt, M.W., Baldridge, K.K., Boatz, J.A., Elbert, S.T., Gordon, M.S., Jensen, J.H., Koseki, S., Matsunaga, N., Nguyen, K.A., Su, S.J., Windus, T.L., Dupuis, M., Montgomerya, J.A.: General atomic and molecular electronic structure system. J. Computational Chemistry **14** (1993) 1347–1363
7. Janssen, C.L., Nielsen, I.M.B., Colvin, M.E.: Encyclopedia of computational chemistry. In Schleyer, P.V.R., Allinger, N.L., Clark, T., Gasteiger, J., Kollman, P.A., III, H.F.S., Scheiner, P.R., eds.: Encyclopedia of Computational Chemistry. John Wiley & Sons, Chichester, UK (1998)
8. Kendall, R.A., Apra, E., Bernholdt, D.E., Bylaska, E.J., Dupuis, M., Fann, G.I., Harrison, R.J., Ju, J.L., Nichols, J.A., Nieplocha, J., Straatsma, T.P., Windus, T.L., Wong A.T.: High performance computational chemistry: An overview of NWChem, a distributed parallel application. Comput. Phys. Commun. **128** (2000) 260–270
9. Dongarra, J., Eijkhout, V.: Self-adapting numerical software for next generation applications. Int. J. High Performance Computing Applications **17** (2003) 125–131 also LAPACK Working Note 157, ICL-UT-02-07.

10. Whaley, R.C., Petitet, A.: Minimizing development and maintenance costs in supporting persistently optimized BLAS. Software: Practice and Experience **35**(2) (February 2005) 101–121
11. Liu, H., Parashar, M.: Enabling self-management of component based high-performance scientific applications. In: Proc. 14th IEEE Int. Symp. on High Performance Distributed Computing, IEEE Computer Society Press (July 2005)
12. Tapus, C., Chung, I.H., Hollingsworth, J.K.: Active Harmony: Towards automated performance tuning. In: Proc. of SC02. (2002)
13. Vetter, J.S., Worley, P.H.: Asserting performance expectations. In: Proc. SC02. (2002)
14. Bramley, R., Gannon, D., Stuckey, T., Villacis, J., Balasubramanian, J., Akman, E., Berg, F., Diwan, S., Govindaraju, M.: The Linear System Analyzer. In: Enabling Technologies for Computational Science. Kluwer (2000)
15. Zhang, K., Damevski, K., Venkatachalapathy, V., Parker, S.: SCIRun2: A CCA framework for high performance computing. In: Proc. 9th Int. Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS 2004), Santa Fe, NM, IEEE Press (April 2004)
16. Houstis, E.N., Catlin, A.C., Rice, J.R., Verykios, V.S., Ramakrishnan, N., Houstis, C.E.: A knowledge/database system for managing performance data and recommending scientific software. ACM Transactions on Mathematical Software **26**(2) (2000) 227–253
17. McCracken, M.O., Snavely, A., Malony, A.: Performance modeling for dynamic algorithm selection. In: Proc. Int. Conf. on Computational Science (ICCS'03). Volume 2660., Berlin, Springer (2003) 749–758
18. Vuduc, R., Demmel, J., Bilmes, J.: Statistical models for empirical search-based performance tuning. Int. J. High Performance Computing Applications **18**(1) (February 2004) 65–94
19. Cortellessa, V., Crnkovic, I., Marinelli, F., Potena, P.: Experimenting the automated selection of COTS components based on cost and system requirements. J. Universal Computer Science **14**(8) (2008) 1228–1256
20. Becker, S., Koziolek, H., Reussner, R.: The Palladio component model for model-driven performance prediction. J. Systems and Software **82**(1) (2009) 3–22
21. Kappler, T., Koziolek, H., Krogmann, K., Reussner, R.: Towards automatic construction of reusable prediction models for component-based performance engineering. In: Proc. Software Engineering. (Feb. 2008)
22. Shende, S., Malony, A.: The TAU parallel performance system. Int. J. High-Perf. Computing Appl., ACTS Collection special issue **20** (Summer 2006) 287–331
23. Taylor, V., Wu, X., Stevens, R.: Prophesy: An infrastructure for performance analysis and modeling of parallel and grid applications. SIGMETRICS Perform. Eval. Rev. **30**(4) (2003) 13–18
24. de Rose, L.A., Reed, D.A.: SvPablo: A multi-language architecture-independent performance analysis system. In: ICPP '99: Proc. 1999 International Conference on Parallel Processing, IEEE Computer Society (1999)
25. Jorba, J., Margalef, T., Luque, E.: Performance analysis of parallel applications with KappaPI2. In: Proc. Parallel Computing: Current and Future Issues of High-End Computing. (2006) 155–162
26. Voss, M.J., Eigemann, R.: High-level adaptive program optimization with adapt. ACM SIGPLAN Notices **36** (2001) 93–102
27. Song, F., Wolf, F., Bhatia, N., Dongarra, J., Moore, S.: An algebra for cross-experiment performance analysis. In: Proc. 2004 International Conference on Parallel Processing (ICPP04), Montreal, Quebec, Canada. (2004) 63–72

28. Huck, K.A., Malony, A.D., Shende, S., Morris, A.: Scalable, automated performance analysis with TAU and PerfExplorer. In: Parallel Computing. (2007)
29. Szyperski, C.: Component Software: Beyond Object-Oriented Programming. ACM Press, New York (1999)
30. Kenny, J.P., Benson, S.J., Alexeev, Y., Sarich, J., Janssen, C.L., McInnes, L.C., Krishnan, M., Nieplocha, J., Jurrus, E., Fahlstrom, C., Windus, T.L.: Component-based integration of chemistry and optimization software. J. Computational Chemistry **25**(14) (2004) 1717–1725
31. Peng, F., Wu, M.S., Sosonkina, M., Bentz, J., Windus, T.L., Gordon, M.S., Kenny, J.P., Janssen, C.L.: Tackling component interoperability in quantum chemistry software. In: Workshop on Component and Framework Technology in High-Performance and Scientific Computing, in conjunction with OOPLSA. (2007)
32. Wu, M.S., Bentz, J., Peng, F., Sosonkina, M., Gordon, M.S., Kendall, R.A.: Integrating performance tools with large-scale scientific software. In: The 8th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC-07), in conjunction with 21st International Parallel and Distributed Processing Symposium (IPDPS-07), Long Beach, California (March 2007)
33. Malony, A., Shende, S., Trebon, N., Ray, J., Armstrong, R., Rasmussen, C., Sottile, M.: Performance technology for parallel and distributed component software. Concurrency and Computation: Practice and Experience **17** (2005) 117–141
34. Huck, K.A., Malony, A.D., Bell, R., Morris, A.: Design and implementation of a parallel performance data management framework. In: Proc. 2005 International Conference on Parallel Processing, ICPP2005, Los Alamitos, CA, USA, IEEE Computer Society (2005) 473–482
35. Witten, I.H., Frank, E.: Data mining: Practical machine learning tools and techniques. http://www.cs.waikato.ac.nz/ ml/weka/ (2005)
36. Allan, B., Armstrong, R., Lefantzi, S., Ray, J., Walsh, E., Wolfe, P.: Ccaffeine – a CCA component framework for parallel computing. `http://www.cca-forum.org/ccafe/` (2003)
37. Møller, C., Plesset, M.S.: Note on an approximation treatment for many-electron systems. Phys. Rev. **46** (1934) 618 – 622
38. U. S. Dept. of Energy: SciDAC Initiative homepage. `http://www.osti.gov/scidac/` (2006)
39. Bernholdt D. (PI): TASCS Center. `http://www.scidac.gov/compsci/TASCS.html`
40. Lucas R. (PI): Performance Engineering Research Institute (PERI). `http://www.peri-scidac.org`