RMOST: A Shared Memory Model for Online Steering*

Daniel Lorenz[1,2], Peter Buchholz[1], Christian Uebing[3], Wolfgang Walkowiak[1], and Roland Wismüller[2]

[1] Experimental Particle Physics, University of Siegen, Germany
[2] Operating Systems and Distributed Systems, University of Siegen, Germany
[3] Center for Information and Media Technology, University of Siegen, Germany
daniel.lorenz@uni-siegen.de

Abstract. Online steering means to visualize the current state of an application which includes application data and/or performance data, and to modify data in the application. Thus, in online steering the application as well as the steering tool must concurrently access and modify the same data at run time. In this paper a new model for online steering is presented which models the mechanism of online steering as access to a distributed shared memory. The integrity requirements of the steered application are analyzed. The integrity can be ensured through an appropriate consistency model. Finally, the online steering system RMOST is presented which is based on the distributed shared memory model and can be used to steer Grid jobs from the High Energy Physics experiment ATLAS.

1 Introduction

In recent times, scientific simulations increased both in complexity and in the amount of data they produce. Often, the simulations run on batch systems in clusters or computational Grids and do not support interactivity during the runtime of the simulation. Online steering of an application enables the visualization of intermediate results, performance data, or other application data, and the invocation of actions, e.g. modification of a parameters by the user at runtime of the job. The user can interactively explore parameter realms, debug his program, or optimize performance. Beause the user sees results earlier, he can evaluate results earlier and react before the job has finished. Thus, online steering accelerates scientific research and saves ressources.

In this work application means the steered program. A steering tool is distinguished from a steering system. A steering tool is the interface to the user which visualizes data and offers the user the possibility to enter commands, e.g. modifications of a parameter. A steering system comprises all extentions to the application, external components, specialized steering tools, and extentions to offline visualization tools to enable steering. For example, a steering system can

* This work is partly funded by the *Bundesministerium für Bildung und Forschung* (BMBF) as part of the German e-Science Initiative (Contract 01AK802E, HEP-CG).

M. Bubak et al. (Eds.): ICCS 2008, Part III, LNCS 5103, pp. 223–232, 2008.
© Springer-Verlag Berlin Heidelberg 2008

comprise steering libraries on the application side, special processes which are needed for the communication or automated decision making, and libraries on the steering tool side to extend an existing visualization.

Until now, various steering systems have been developed [1,2,3,4,5,6,7] for supporting the scientist with interactive control over his simulation. Existing systems provide means to retrieve data from the application and invoke actions in the remote application. Typically, the application is instrumented with calls to a steering library to enable the sending of data to a remote visualizer, or to apply commands from the user. The steering system manages the data transport to a customized user interface. One of the reasons why steering systems are not used is the required effort to instrument a legacy application for steering.

In this paper another approach is used, which views steering similar to distributed shared memory (DSM). Any steering is basically the change of state of an application and a state change corresponds to a change in memory. Thus, steering can be modeled as a case for DSM because all steering actions can be reduced to memory access operations. This approach simplifies the application of steering systems to existing software and improves the efficiency of steering.

If a data object is modified in a running application without any synchronization with the execution of the applications, severe errors may occur.

In these cases the integrity of the data in the application is broken. To protect the integrity, rules are needed which define the order of access operations on the shared data. The necessary rules define a consistency model. Though various steering tools exists, until now no consistency model for online steering exists. In most cases the integrity problem is not addressed or left to the user.

Based on the DSM-based steering model, the new online steering system RMOST (Result Monitoring and Online Steering Tool) [8,9] was developed. The advantages for the user are the ease of use of a DSM-like approach and the build-in consistency guarantees.

2 Formalism for the DSM Based Model for Online Steering

In online steering, the application and the steering tool access both the same data. If the steering tool and the application run in the same address space, it is a trivial task. But if visualization and simulation have different address spaces, e.g. if they are located on different machines, a mechanism to access the remote data is needed. Thus, online steering can be modeled as DSM. The advantages of a DSM model are that the complexity of distributed data is hidden from the user of the steering system, and it looks like accessing only local data for the steering tool and the application. The steering system completely handles the communication and it supports the programmer with the consistency guarantees to maintain data integrity.

In the DSM based model of online steering, two kinds of processes exists with different roles and properties. Firstly, n application processes $p_1, ..., p_n$ exist. The application may synchronize $p_1, ..., p_n$ with any mechanism, e.g. MPI, or shared

memory. However, the synchronization within the application is out of scope of this work. Secondly, m steering processes $p_{n+1}, ...p_{n+m}$ exist, each representing a steerer in a collaborative environment. The data objects which can be visualized or steered reside in the distributed shared memory. Each data object o has a home location $H(o) \in p_1, ..., p_n$ which is one of the application processes. The steering processes are not chosen for home locations, because the steerers may detach and thus causing the home location to be inaccessible.

Three kinds of memory operations exist: read operations r, write operations w, and synchronization operations s. Read and write operations are denoted as $o(p, x, v)$ where $o \in \{w, r\}$ specifies the operation type, p is the process that perform the operation, x is the memory location, and v is the value that is written or read. Synchronization operations are denoted as $s(p, x)$. A process p_i is viewed as a sequence of memory operations $S_i = \{o_1, o_2, ...\}$ with $o_i \in \{w, r, s\}$. A process *sees* a write operation w if a current read operation would return the value written by w. A write operation w is *visible* to a process p if p can see w.

Each application process p_i is associated with a logical clock T_i, which indicates the progress of the process. T_i in incremented when p_i release an update to the distributed shared memory, or when p_1 sees an update of another process. Thus, synchronization operations imply clock incrementations. An *epoch* is the interval between two consecutive clock incrementations. The furthest common logical time $T_{min} = \min(T_1, ..., T_n)$ is the minimum time of all application processes. The furthest time $T_{max} = \max(T_1, ..., T_n)$ is the maximum time of all application processes.

3 Data Integrity

Data integrity is an important prerequisite to obtain correct results from the application. This means a steering system should ensure that the displayed data is consistent in itself, and any modifications must preserve the integrity of the data within the application. In this section, the effects that might affect the integrity are analyzed which lead to two integrity conditions. The first one is the *inner-process* condition, and the second one is the *inter-process* condition.

3.1 The Inner-Process Condition

The inner-process condition requires that the data in the application must not be modified externally during certain operation intervals, and that the write operations of the application to shared objects become only visible if the data is in a well-defined state. For example, assume one formula is computed where one variable x appears at different places in the formula. The result can only be correct if the value of x stays the same during the whole computation. Another case could be a numerical n-body simulation. While it is allowed to modify parameters between each simulated time step, the value should stay the same inside each simulated time step.

Also the modifications of the application to shared data should become visible only at well defined places. Imagine several properties of different input objects

are computed. If the object is visible and displayed after the computation of the first few properties while the other properties stem from another input, the displayed result is propably incorrect and can be misleading. Thus, to preserve the inner-process condition, changes of the application must only become visible at well defined points, and changes by the steerer must only be applied by the application at well-defined *synchronization* points. Typically, one epoch is bounded by two synchronization points, which implies that synchronization points match the incrementations of the logical clock.

3.2 The Inter-Process Condition

The inter-process condition considers the different progress of different processes. Firstly, it requires that write operations of the steering processes must be seen in all processes at the same time step. Secondly, values of displayed data objects must stem from the same epoch.

For example, suppose a parallel simulation iterates over several time steps and each process computes a part of the overall result. If changing a boundary parameter, one would like to change this parameter at all processes in the same epoch. If a steerer changes the value of the parameter in the DSM, the system must ensure that the modification is viewed by all processes at the same epoch.

Another case occurs if a steerer wants to display a distributed object which is modified by several processes, and each process computes a part of the whole object. The steerer must only see the writes of all processes up to T_{min} to retrieve an internally consistent data set. Writes of processes that have proceeded further ahead must not be visible to the steerer to provide a well-defined display of intermediate results.

4 Consistency Models

To ensure the integrity of the data in online steering, each process must view access operations to the shared memory according to certain rules. For each given set of access operations, a consistency model is defined through the possible orders in which each process is allowed to see the memory accesses [10]. Thus, a consistency model can be used to maintain data integrity. In this section, consistency models are evaluated which fulfill the requirements for data integrity in online steering. One consistency model will not satisfy all cases, because not all data objects require both integrity requirements analyzed in Sec. 3. Some data objects have no integrity conditions and can be treated completely asynchronous, some data objects have only the inner-process condition, and some data objects require both conditions. Thus, different consistency models are appropriate to each of these cases. The case that data objects have only the inter-process condition is not considered, because the inter-process condition implies the existence of epoches. The transition points between two epoches define the synchronization points where values may be read or modified.

4.1 Consistency for the Inner-Process Condition

The inner-process condition allows the application and distribution of updates only at special synchronization points. The desired existence of special synchronization points leads to a consistency model which is similar to weak consistency [11]. Two possibilities exist which behave different in the following case: Let p_1 be an application process and let p_2 be a steerer process that viewed the accesses $w(p_1, x, 1)$, and $s(p_1)$. Now, p_2 executes $w(p_2, x, 2)$, and $r(p_2, x, ?)$ before it sees another $s(p_1)$. Which value should $r(p_2, x, ?)$ return?

1. $r(p_2, x, 1)$ returns the current value of the application. The newly written value is not seen until the next synchronization operation. This model delays the execution of the write operation after the next synchronization operation, thus it is called *delayed weak consistency*. This consistency model displays always a consistent set of values from the application, but it has the effect that a read operation at the steerer may not return the value written by the previous write operation. This behavior can be interpreted as display of results. An advantage of this model is that it does not require a sequential order of the synchronization points.
2. $r(p_2, x, 2)$ returns the value recently written by the same process. This leads to weak consistency with the modification that updates are applied exactly at the next synchronization operation, instead of *latest* at the next synchronization operation. In this case a read operation of the steerer may return a value that is not consistent with the results from the application. The displayed data equals the value the application sees when it enters the next epoch. It can be interpreted as display of the configuration.

Interestingly, the sequential consistency [12] is too strong for the inner-process condition. In most DSM systems, the usage of relaxed consistency models is driven by the better performance of the relaxed models compared to strong consistency models, but the programmer wants his program to behave like sequential consistency [12] would be used. In the case of online steering, strong consistency would not provide the desired behavior. In the example shown above both cases violate the rules of sequential consistency. With delayed weak consistency a read does not return the value of the most recent write, and with weak consistency p_1 and p_2 view write operations in different orders.

If synchronization operations are not global but only for one or a few data objects, release like consistency models [13] can be derived. But this reduces the advantage of a simple instrumentation, because it requires more detailed information about which data is updated at each synchronization point.

4.2 Consistency for Both Conditions

In this case it must be ensured that the steerers retrieve all values from the same epoch, and all application processes apply all modifications at the same epoch. At every given time, each epoch can be assigned to one of the following three groups:

- The *past* are those epoches T that are finished by all application processes: $T < T_{min}$.
- The *future* are those epoches T that are not yet entered by any application process: $T > T_{max}$.
- The *presence* are the epoches T that do neither belong to the past nor to the future: $T \in [T_{min}, T_{max}]$.

Each write operation w is tagged with a time stamp $T(w)$. Write operations of an application process p will be tagged with the timestamp of the process $T(w) = T(p)$. Write operations of a steering processes will be tagged with $T_{max} + 1$. Thus, each data object has a schedule of values assigned to it. A read operation of data object x by a steering process will always return the most recent value v from the past. Read operations of an application process p at time step $T(p)$ will always return the most recent value v from the viewpoint of the process.

This consistency model is called *schedule consistency*. Steerers can only write to the future and read from the past. It has the effect, that modifications are not seen immediately, but after a delay which depends on the length of the presence. The delayed weak consistency is a special case of the schedule consistency with the presence comprising only one epoch. Formally, this effect is caused by an reordering of write and read operations in the steerer processes. Writes that occur before a read in program order may be seen later than the read.

4.3 Consistency with No Integrity Conditions

Beside parameters or results which probably have the inner-process or inter-process condition, data objects with a producer-consumer access pattern exists which require none of the integrity conditions. These data object have one producer, which is the only process writing to this data object, and one or more consumer processes who read this data object. For example, processor load or other monitoring data has neither the inner-process nor the inter-process condition. For those data the update intervals or delays implied by the weak or schedule consistency may be inappropriate. These data objects are independent from other data objects by definition, thus Pipelined RAM consistency [14] should be sufficient. Pipelined RAM consistency ensures that all processes view the writes of a process p in the order they are executed by p.

5 Implementation in RMOST

RMOST (Result Monitoring and Online Steering Tool)[1] [8,9] is an online steering system for Grid Jobs of the High Energy Physics (HEP) experiment ATLAS [15]. It consists of an application independent implementation of the presented DSM approach for online steering, and a thin integration layer into the ATLAS software. Through the DSM-based approach it is possible to enable steering of Grid

[1] RMOST can be downloaded from http://hep.physik.uni-siegen.de/grid/rmost

Jobs in the ATLAS experiment without modification of the source code. Currently, only sequential applications with one steerer are supported. Its architecture consists of four main layers:

1. The communication layer realizes a communication channel between the application and the steering tool. The Grid communication channel of RMOST is described in [16].
2. The data consistency layer implements a DSM system with the consistency models described in Sec. 4.
3. The data processing layer is a place holder for any data processing performed by the steering system like filtering, or automated evaluation.
4. The data access layer provides tools for data access. For example, in RMOST a preloaded library replaces standard library calls in order to observe file accesses. Another (not yet implemented) possibility is to monitor method calls by modifying a classes' virtual table.

5.1 Data Consistency Layer

The data consistency layer provides a framework for several consistency protocols implementing different consistency models. The framework consists of the registry, the manager, and an interface for consistency protocols.

The registry contains for every steerable data object its name, the used protocol, and the access methods of the local copy. If several processes register data with same name, these objects are considered as local copies of the same value. The manager handles all communication in a separate thread. Asynchronous messages are immediately forwarded to the appropriate protocol, while synchronized message types are buffered until the next synchronization operation.

Consistency protocols can send messages via the manager and are called on every synchronization point, when a message is received for it, or if a data object is accessed which uses this protocol. Currently, for delayed weak consistency, pipelined RAM consistency, and blockwise delayed weak consistency both an invalidate and an update protocol are implemented. As example, the update protocol for the weak and for the delayed weak consistency are explained:

If a process uses the update protocol for the weak consistency and a write occurs, it sets a modification flag for this data object. At the next synchronization point, the process sends updates of all data objects whose modification flags have been set. If a process receives an update, it is buffered until the next synchronization point. At this point the new value is applied. If an update was received and the local modification flag of this data object is set, too, the value from the steering process has priority. For weak consistency the synchronization points must be ordered sequentially. Thus, a process must obtain a synchronization lock before it can execute a synchronization point.

If the application uses delayed weak consistency, it performs the same actions as with weak consistency except that it does not obtain the synchronization lock. If the steering tool performs a write operation, the new value is buffered, but not yet applied locally. At the next synchronization point an update is

send to the application. If the application receives an update, it applies it at its next synchronization point. Afterwards, it sends an acknowledgment back to the steering tool. The steering tool applies the new value at the next synchronization point after receiving the acknowledgment.

6 Application of RMOST

The ATLAS [15] experiment is performed at the Large Hadron Collider (LHC) at CERN. Beside many others, the most prominent goal of the ATLAS experiment is to find the Higgs particle which is responsible for the masses of particles.

The experiment software framework Athena [17] was created for the computation of the data and is commonly used in the HEP community. The processed data consists of collision events which can be computed independently. In general, the desired results are statistics over several thousands events.

The Athena framework [17] provides different components which can be plugged together by the user through a so called job options file. Furthermore, Athena can be extended with customized components contained in a shared library. The different components can be categorized into several basic classes. The two important classes for the implementation of RMOST are algorithms and services. The core of an Athena job is a list of algorithms which are executed for each event. Services provide functionality to other components.

The ROOT toolkit [18] is commonly used for offline visualization of physical results. It provides an interface to extend ROOT with new classes which are located in a shared library and loaded dynamically. Modifications and recompilations of the ROOT toolkit and the Athena framework to enable steering are hardly accepted by the HEP community. Thus, for the integration of RMOST in the Athena framework a new algorithm `RM_Spy` was developed which can be applied to the Grid job by editing the job options file. `RM_Spy` enables the steering of the job execution, monitoring of intermediate results in the output files, and modification of the job options file. The steering API is encapsulated by a new Athena service `RM_SteeringSvc`. Thus, steering of Athena jobs is enabled without modification of the source code of existing components. Other components can be extended with customized steering features by using the `RM_SteeringSvc`.

Steering can be made available to ROOT by dynamically loading an extra library with interface classes for ROOT to RMOST. It allows to modify steerable parameters, or view progress information from the job. Through preloading of the RMOST file access library, the steering system intercepts file accesses and fetches or updates the according parts of the file. Thus, the existing offline visualization in ROOT can be used for online monitoring of intermediate results and steering without modifications of the source code.

7 Related Work

Until now, no general DSM-based model for steering exists. However, some steering tools provide tools to support the user to maintain the integrity of the data.

Closest to this work is the Pathfinder [5] steering system. Steering actions and the program's execution are both viewed in terms of atomic transactions. They address the issue to consistently apply steering actions to a parallel message passing program. A steering action is consistent if it is applied in a consistent snapshot of the parallel program. An algorithm is presented which detects inconsistent steering actions. The mayor issue is to define points in a parallel application where steering actions can be consistently applied. As result a global ordering of all transactions exists, which leads to sequential consistency.

CUMULVS [2,19] is a steering tool which allows to make checkpoints of a parallel program. An algorithm is presented to capture distributed data objects consistently by stopping processes that have already processed ahead until all processes reached an equal progress. While this algorithm is similar to the presented schedule consistency, CUMULVS has no DSM-based model for steering.

EPSN [20] requires a description of the structure of the application. For each steerable data object, areas are defined where the data object may be read or changed. The source code of the application must be instrumented with markers to the abstract structure. VASE [6] follows similar principles. The integrity problem is brought to an abstract level which can simplify the problem for the user. However, the decision where a data object may be accessed without disrupting integrity stays with the user. Both have no DSM-based approach for steering.

In RealityGrid [1] a client/server based steering system was developed. The steering library only informs the application on events which must be handled by the user. The user may use predefined library calls to react on events, but a DSM like mechanism does not exist. The steering actions are performed in a single steering library call to reduce the effort of instrumentation. Because events are processed in a single function, by default, weak consistency is implicitly realized.

8 Conclusions and Future Work

The data integrity of an application can be destroyed through online steering. Two major conditions that ensure data integrity are identified, the inner-process condition and the inter-process condition. Online steering is viewed as a access to distributed shared memory. The integrity of the data can be maintained if the steering systems provide certain consistency guarantees. This allows to easily apply steering to existing legacy codes. In the case of the ATLAS experiment it was possible to enable offline legacy codes for online steering without changing existing codes by just adding components to the framework. Furthermore, offline visualization tools could be used for online visualization.

The necessary conditions may vary between different objects of the same application, thus a steering system should support a number of consistency models. First measurements show, that invalidate protocols effectively avoid overload on the network, which happens in stream-based steering tools. On the other hand, update protocols are faster for small amounts of data. We are currently working on an automatic selection between update or invalidate protocols that dynamically adapts to the environment and optimizes the performance.

References

1. Jha, S., et al.: A computational steering API for scientific Grid applications: Design, implementation and lessons. In: Workshop on Grid Application Programming Interfaces (September 2004)
2. Geist, G.A., et al.: CUMULVS: Providing fault-tolerance, visualization and steering of parallel applications. Int. J. of High Performance Computing Applications 11(3), 224–236 (1997)
3. Vetter, J.S., et al.: High performance computational steering of physical simulations. In: Proc. of the 11th Int. Symp. on Parallel Processing, pp. 128–134. IEEE, Los Alamitos (1999)
4. Ribler, R.L., et al.: The Autopilot performance-directed adaptive control system. Future Generation Computer Systems 18(1), 175–187 (2001)
5. Hart, D., et al.: Consistency considerations in the interactive steering of computations. Int. J. of Parallel and Distributed Syst. and Networks 2(3), 171–179 (1999)
6. Brunner, J.D., et al.: VASE: the visualization and application steering environment. In: Proc. 1993 ACM/IEEE conference on Supercomputing, pp. 560–569 (1993)
7. Brodlie, K., et al.: Visualization in grid computing environments. In: Proc. of IEEE Visualization 2004, pp. 155–162 (2004)
8. Lorenz, D., et al.: Online steering of HEP Grid applications. In: Proc. Cracow Grid Workshop 2006, Cracow, Poland, Academic Computer Centre CYFRONET AGH, pp. 191–198 (2007)
9. RMOST webpage, `http://www.hep.physik.uni-siegen.de/grid/rmost`
10. Steinke, R.C., Nutt, G.J.: A unified theory of shared memory consistency. J. of the ACM 51, 800–849 (2004)
11. Dubois, M., et al.: Memory access buffering in multiprocessors. In: ISCA 1986: Proc. of the 13th annual int. symp. on Computer architecture, pp. 434–442. IEEE Computer Society Press, Los Alamitos (1986)
12. Lamport, L.: How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE Trans. Comp. C-28(9), 690–691 (1979)
13. Gharachorloo, K., et al.: Memory consistency and event ordering in scalable shared-memory multiprocessors. In: 25 Years ISCA: Retrospectives and Reprints, pp. 376–387 (1998)
14. Lipton, R.J., Sandberg, J.S.: PRAM: A scaleable shared memory. Technical Report CS-TR-180-88, Princeton University (September 1988)
15. The ATLAS Experiment, `http://atlasexperiment.org`
16. Lorenz, D., et al.: Secure connections for computational steering of Grid jobs. In: Proc. 16th Euromicro Int. Conf. on Parallel, Distributed and network-based Processing, Toulouse, France, pp. 209–217. IEEE, Los Alamitos (2008)
17. Athena Developer Guide, `http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/z0_obsolete/architecture/General/Tech.Doc/Manual/AthenaDeveloperGuide.pdf`
18. Brun, R., et al.: ROOT - an object oriented data analysis framework. In: Proc. AIHENP 1996 Workshop. Number A 389 in Nuclear Instruments and Methods in Physics research, pp. 81–86 (1997) (1996)
19. Papadopoulos, P.M., et al.: CUMULVS: Extending a generic steering and visualization middleware for application fault-tolerance. In: Proc. 31st Hawaii Int. Conf. on System Sciences (HICSS-31) (January 1998)
20. Esnard, A., Dussere, M., Coulaud, O.: A time-coherent model for the steering of parallel simulations. In: Danelutto, M., Vanneschi, M., Laforenza, D. (eds.) Euro-Par 2004. LNCS, vol. 3149, pp. 90–97. Springer, Heidelberg (2004)