On Utilizing Experiment Data Repository for Performance Analysis of Parallel Applications ⋆

Hong-Linh Truong, Thomas Fahringer

Institute for Software Science, University of Vienna
Liechtensteinstr. 22, A-1090 Vienna, Austria
{truong,tf}@par.univie.ac.at

Abstract. Performance data usually must be archived for various performance analysis and optimization tasks such as multi-experiment analysis, performance comparison, automated performance diagnosis. However, little effort has been done to employ data repositories to organize and store performance data. This lack of systematic organization of data has hindered several aspects of performance analysis tools such as performance comparison, performance data sharing and tools integration. In this paper we describe our approach to exploit a relational-based experiment data repository in SCALEA which is a performance instrumentation, measurement, analysis and visualization tool for parallel programs. We present the design and use of SCALEA's experiment data repository which is employed to store information about performance experiments including application, source code, machine information and performance data. Performance results are associated with experiments, source code and machine information. SCALEA is able to offer search and filter capabilities, to support multi-experiment analysis as well as to provide well-defined interfaces for accessing the data repository and leveraging the performance data sharing and tools integration.

1 Introduction

Collecting and archiving performance data are important tasks required by various performance analysis and optimization processes such as multi-experiment analysis, performance comparison and automated performance diagnosis. However, little effort has been done to employ data repositories to organize and store performance data. This lack of systematic organization of data has hindered several aspects of performance analysis tools. For example, users commonly create their own performance collections, extract performance data and use external tools to compare performance outcome of several experiments manually. Moreover, different performance tools employ different performance data formats and they lack well-defined interfaces for accessing the data. As a result, the collaboration among performance tools and high-level tools is hampered.

Utilizing a data repository and providing well-known interfaces to access data in the repository can help to overcome the abovementioned limitations. We can

⋆ This research is partially supported by the Austrian Science Fund as part of the Aurora Project under contract SFBF1104.

structure the data associated with performance experiments thus performance results can always be associated with their source codes and machine description on which the experiment has been taken. Based on that, any other performance tool can store its performance data for a given application to the same repository thus providing a large potential to enable more sophisticated performance analysis. And then, any other tools or system software can easily access the performance data through a well-defined interface. To do so, we have investigated and exploited a relational-based experiment data repository in SCALEA [12, 11] which is a performance instrumentation, measurement, analysis and visualization tool for parallel programs.

In this paper, we present the design and use of SCALEA's experiment data repository which is employed to store performance data and information about performance experiment which alleviates the association of performance information with experiments and source code. SCALEA's experiment data repository has been implemented with relational database, SQL and accessed through interfaces based on JDBC. We demonstrate significant achievements gained when exploiting this data repository such as the capabilities to support search and multi-experiment analysis, to facilitate and leverage performance data sharing and collaboration among different tools. We also discuss other directions on utilizing performance data repository for performance analysis.

The rest of this paper is organized as follows: Section 2 details the SCALEA's experiment data repository. We then illustrate achievements gained from the use of the experiment data repository in Section 3. We discuss other directions to utilize the experiment data repository in Section 4. The related work is presented in Section 5 followed by the conclusion and future work in Section 6.

2 Experiment Data Repository

2.1 Experiment-related data

Figure 1 shows the structure of the data stored in SCALEA's experiment data repository. An *experiment* refers to a sequential or parallel execution of a program on a given target architecture. Every experiment is described by *experiment-related data*, which includes information about the application code, the part of a machine on which the code has been executed, and performance information. An application (program) may have a number of code versions, each of them consists of a set of source files and is associated with one or several experiments. Every source file has one or several static code regions (ranging from entire program units to single statements), uniquely specified by their positions – start/end line and column – where the region begins and ends in the source file.

Experiments are associated with virtual machines on which they have been taken. The virtual machine is a collection of physical machines to execute the experiment; it is described as a set of *computational nodes* (e.g. single-processor systems, SMP nodes) of clusters. A *Cluster* is a group of computational nodes (physical machines) connected by specific networks. Computational nodes in the same cluster have the same physical configuration. Note that this structure is

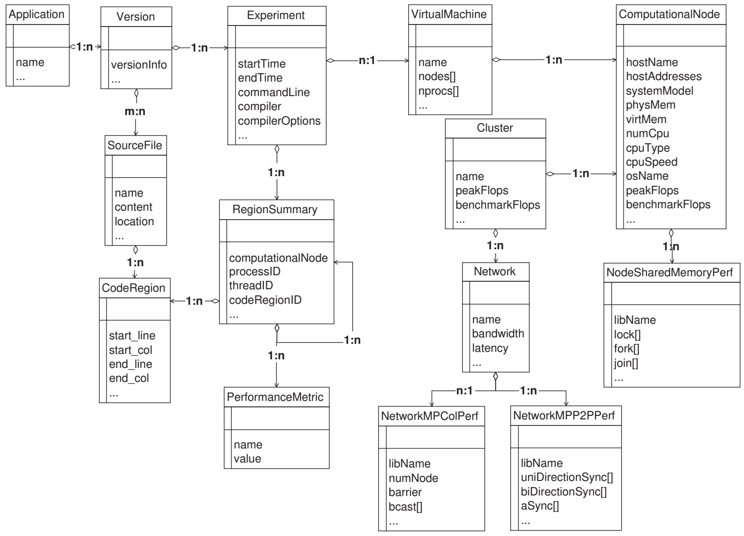

Fig. 1. SCALEA's Experiment-Related Data Model

still suitable for a network of workstations as workstations can be classified into groups of machines having the same configuration. Specific information of physical machines such as memory capacity, peak FLOPS is measured and stored in the data repository. In addition, for each computational node, performance characteristics of shared memory operations (e.g. lock, barrier, fork/join thread) are benchmarked and stored in *NodeSharedMemoryPerf*. Similarly, for each network of a cluster, performance characteristics of message passing model are also benchmarked and stored in *NetworkMPColPef* and *NetworkMPP2PPerf* for collective and point-to-point operations, respectively.

A *region summary* refers to the performance information collected for a given code region on a specific *processing unit* (consisting of computational node, process, thread). The region summaries are associated with performance metrics that comprise performance overheads, timing information, and hardware parameters. A region summary has a link to a parent region summary; this link reflects the calling relationship between the two regions recorded in the dynamic code region call graph [12].

2.2 Implementation Overview

Figure 2 depicts components that interact with the experiment data repository. The *post-processing* is used to store source programs and the instrumentation description file [12] generated by SCALEA instrumentation system [11] into the data repository. In addition, it filters raw performance data collected for each experiment and stores the filtered data into the repository.

The *overhead analyzer* performs the overhead analysis according to the overhead classification [11] based on filtered data in the repository (or trace/profile files). The resulting overhead is then stored into the data repository.

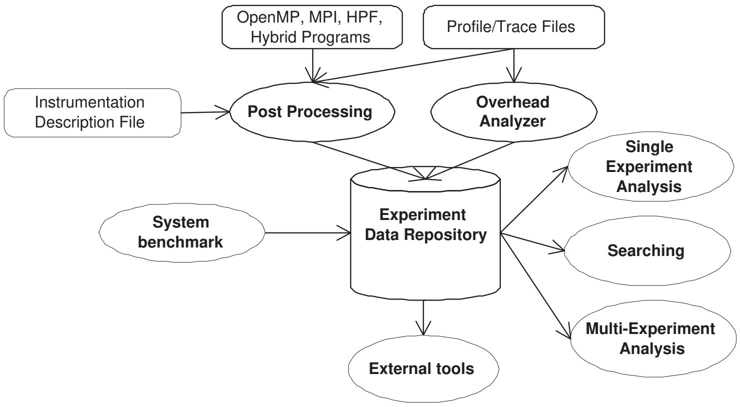

Fig. 2. Components interacting with SCALEA's Experiment Data Repository

The *system benchmark* is used to collect system information (memory, harddisk, etc), to determine specific information (e.g. overhead of probes, time to access a lock), and to perform benchmarks for every target machine of interest. By using available MPI and OpenMP benchmarks, reference values of system performance are obtained for both message passing (e.g. blocking send/receive, collective) and shared memory operations (e.g. lock, barrier) with various networks (e.g. Fast-Ethernet, Myrinet), libraries (e.g. MPICH, PGI OpenMP) on different architectures (e.g. Linux x86, Sun Sparc). The obtained results aim to assist the correlation analysis between application- and system-specific metrics.

Based on data availability in the repository, various analyses can be conducted such as single- and multi-experiment analysis, searching. Moreover, data can be exported into XML format which further facilitates accessing performance information by other tools (e.g. compilers or runtime systems) and applications.

The experiment data repository is implemented with PostgreSQL [8] which is a relational database system supported on many platforms. All components interacting with the data repository are written in Java and the connection realizing between these components with the database is powered by JDBC.

2.3 Experiment-related Data APIs

Each table shown in Fig. 1 is associated with a Java class. In addition, we define two classes *ProcessingUnit* and *ExperimentData*. The first class is used to describe the *processing unit* where the code region is executed; a process unit consists of information about *computational node, process, thread*. The latter implements interfaces used to access experiment data. Bellow, we just highlight some classes with few selected data members and methods:

```
public class PerformanceMetric {
   public String metricName;
   public Object metricValue;
   ...
}
```

```
public class ProcessingUnit {
   ...
   public ProcessingUnit(String node,int process, int thread) {...}
}
public class RegionSummary {
   ...
   public PerformanceMetric[] getMetrics(){...}
   public PerformanceMetric getMetric(String metricName){...}
}
public class ExperimentData {
   DatabaseConnection connection;
   ...
   public ProcessingUnit[] getProcessingUnits(Experiment e){...}
   public RegionSummary[] getRegionSummaries(CodeRegion cr, Experiment e){...}
   public RegionSummary getRegionSummary(CodeRegion cr, ProcessingUnit pu,
           , Experiment e) {...}
   ...
}
```

Based on the well-defined APIs, external tools can easily access data in the repository, using the data for their own purpose. For example, a tool can compute $\frac{identified\ overhead}{total\ execution\ time}$ ratio for code region 1 in computational node *gsr1.vcpc.univie.ac.at*, process *1*, thread *0* of experiment 1 based on identified overhead (denoted by *oall_ident*) and total execution time (denoted by *wtime*) as follows:

```
CodeRegion cr = new CodeRegion(``region1'');
Experiment e = new Experiment(``experiment1'');
ProcessingUnit pu = new ProcessingUnit(``gsr1.vcpc.ac.at'',1,0);
ExperimentData ed = new ExperimentData(new DatabaseConnection(...));
RegionSummary rs = ed.getRegionSummary (cr,pu,e);
PerformanceMetric overhead=rs.getMetric(''oall_ident'');
PerformanceMetric wtime =rs.getMetric(``wtime'');
double overheadRatio=((Double)overhead.metricValue).doubleValue()/
             ((Double)wtime.metricValue).doubleValue();
```

3 Achievements of Using Experiment Data Repository

3.1 Search and Filter Capabilities

Most existing performance tools lack basic search and filter capabilities. Commonly, the tools allow the user to browse code regions and associated performance metrics through various views (e.g. process time-lines with zooming and scrolling, histograms of state durations and message data [14, 3]) of performance data. Those views are crucial but mostly require all data to be loaded into the memory, eventually making the tools in-scalable. Moreover, the user has difficulty to find out the occurrence of events with interesting criteria of performance metrics, e.g. code regions with overhead of data movement [11] larger than 50% of total execution time. Search of performance data will allow the user to quickly identify interesting code regions. Filtering performance data being visualized helps to increase the scalability of performance tools. Utilizing a data repository allows to power the archive, to facilitate search and filter with great flexibility and robustness based on SQL language and to minimize the implementation's cost.

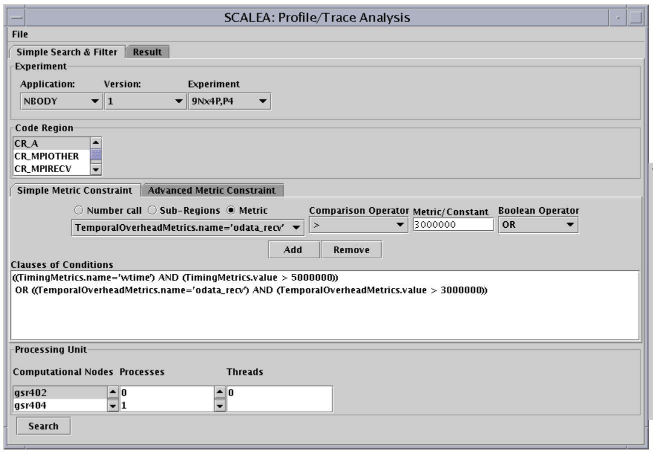

Fig. 3. Interface for Search and Filter in SCALEA

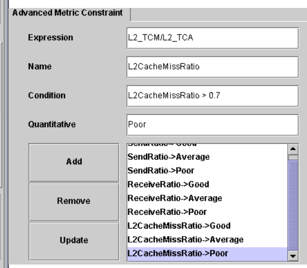

Fig. 4. Specify complex performance conditions

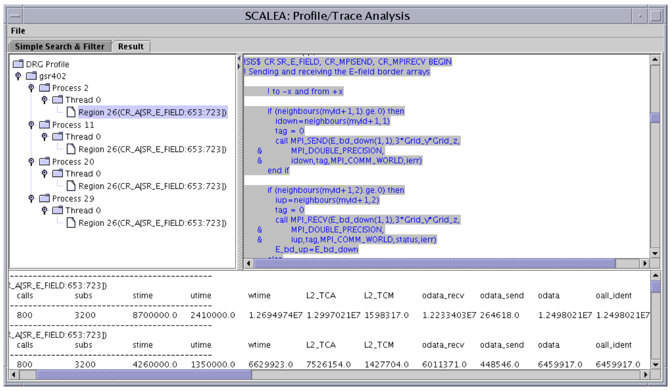

Fig. 5. Results of Performance Search

Figure 3 presents the interface for search and filter in SCALEA. The user can select any experiment for searching code regions under selection criteria. For each experiment, the user can choose *code region types* (e.g. send/receive,OpenMP loop), specify *metric constraints* based on performance metrics (timing, hardware parameter, overhead) and opt the *processing unit* (computational nodes, processes, threads) on which the code regions are executed. Metric constraints can be made in a simple way by forming clauses of selection conditions based on available performance metrics (see Fig. 3). They can also be constructed by selecting quantitative characteristics (Figure 4). For instance, the user may define characteristics for L2 cache miss ratio by expressing L2 cache miss ratio as $r_{L2\ cache\ miss\ ratio} = \frac{L2\ cache\ misses}{L2\ cache\ accesses}$ and then discretizing the L2 cache miss ratio as follows: *good* if $r_{L2\ cache\ miss\ ratio} \leq 0.3$, *average* if $0.3 < r_{L2\ cache\ miss\ ratio}$

< 0.7, and *poor* if $r_{L2\ cache\ miss\ ratio} \geq 0.7$. These quantitative characteristics can be stored into the experiment data repository for later use. SCALEA will transfer user-specified conditions into SQL language, perform the search and return the result. For example, the searching result based on the conditions in Fig. 3 is shown in Fig. 5. In the top-left window, the code region summaries met the search conditions are visualized. By clicking into a code region summary, the source code and corresponding performance metrics of the code region summary will be shown in the top-right and bottom window, respectively.

3.2 Multi-Experiment Analysis

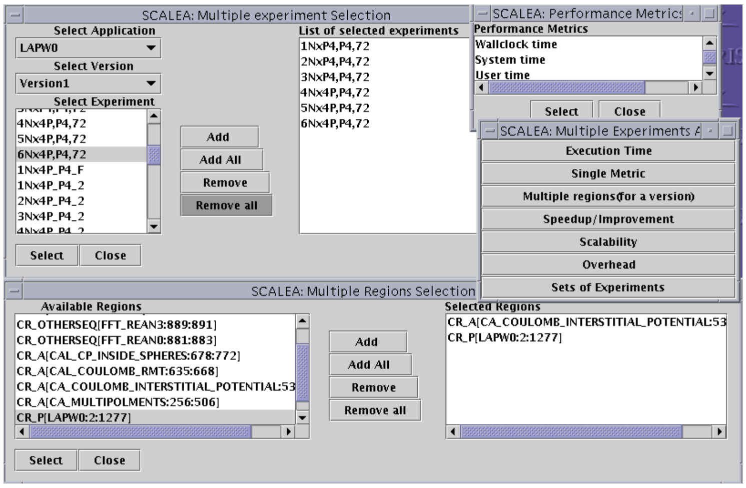

Fig. 6. Interface for Multi-Experiment Analysis

Most existing performance tools [5, 3, 13] investigate the performance for individual experiments one at a time. To archive experiment data in the repository, SCALEA goes beyond this limitation by supporting multi-experiment analysis. The user can select several experiments, code regions and performance metrics of interest of which associated data are stored in the data repository (see Figure 6). The outcome of every selected code regions and metrics is then analyzed and visualized for all experiments. SCALEA's multi-experiment analysis supports:

- **performance comparison for different sets of experiments**: The user can analyze the overall execution of the application across different sets of experiments; experiments in a set are grouped based on their properties (e.g. problem sizes, communication libraries, platforms).
- **overhead analysis for multi-experiment**: Various sources of performance overheads across experiments can be examined.
- **study parallel speedup and parallel efficiency at both program and code region level**: Commonly, those metrics are applied only at the level of the entire program. SCALEA, however, supports to examine the scalability at both program and code region level.

3.3 Data Sharing and Tools Integration

One of the key reasons for utilizing the data repository is the need to support data sharing and tools integration. Via well-defined interfaces, other tools can retrieve data from the experiment data repository and use the data for their own purpose. For example, AKSUM [2] which is a high-level semi-automatic performance bottleneck analysis has employed SCALEA to instrument user's programs, to measure and analyze performance overheads of code regions of the programs. AKSUM then accesses performance data (overheads, timing, hardware parameters) of code regions on the data repository, computes performance properties [1] and conducts the high-level bottleneck analysis based on these properties. In another case, the PerformanceProphet [7] which supports performance modeling and prediction on cluster architectures has used data stored in the experiment data repository to build the cost functions for the application model represented in UML forms. The model is then simulated in order to predict the execution behaviour of parallel programs.

Performance data can also be exported into XML format so that it can easily be transfered and processed by other tools. For example, performance data of *code region 1* can be saved in XML format as follows:

```
<coderegion id="1">
        <metric name=''wtime'' value=''1.09039995E8'' />
        <metric name=''odata_send'' value=''2986000.0'' />
        <metric name=''odata_recv'' value=''5.6923546E7'' />
</coderegion>
```

where each performance metric of the region is represented as a tuple (*name,value*). Note that *wtime* (stands for wall clock time), *odata_send* (overhead of send operations), *odata_recv* (overhead of recv operations) are unique performance metric names described in a *performance metric catalog* which holds information of performance metrics (e.g. unique metric name, data type, well-defined meaning) supported by SCALEA.

4 Further Directions

In this section, we discuss further directions on utilizing the experiment data repository for performance analysis:

- **A query language for performance data** can be designed to support ad hoc and interactive searching and filtering for occurrence of events with criteria of performance metrics and/or performance problems in order to facilitate flexible and efficient discovery of interesting performance information. Such a language can be implemented on top of search facilities provided by database systems and be based on performance property language [1].
- **Automatic scalable analysis techniques** such as decision trees, rule associations, clusters, classifications should be exploited to discover the knowledge of performance information on the repository. These techniques are particularly useful for executing very complex queries on non-main-memory

data. However, currently these techniques are rarely used in performance analysis due to lack of systematic organization of performance data.
- **Providing standardization APIs for acquiring and exploiting performance data** is one of the keys to bring the simplicity, efficiency and success to the collaboration among tools. The well-defined APIs should be independent on the internal data representation and organization of each tool but based on an agreement of well-defined semantics.

5 Related Work

Significant works on performance analysis have been done by Paradyn [10], TAU [5], VAMPIR [3], EXPERT [13], etc. SCALEA differs from these approaches by storing experiment-related data to a data repository, and by supporting also multi-experiment performance analysis.

In [4], information about each experiment is stored in a Program Event and techniques for comparison between experiments are done automatically. A prototype of Program Event has been implemented, however, the lack of capability to export and share performance data has hindered external tools from using and exploiting data in Program Events.

Prophesy [9] provides a repository to store performance data for performing the automatic generation of performance models. Data measured and analyzed by SCALEA can be used by Prophesy for modeling systems.

USRA Tool family [6] collects and combines information of parallel programs from various sources at the level of subroutines and loops. Information is stored in flat files which can further be saved in a format understood by spreadsheet programs. SCALEA's repository provides a better infrastructure for storing, querying and exporting performance data with a relational database system.

APART proposes the performance-related data specification [1] which stimulates our experiment-related data model. Besides performance-related data, we also provide system-related data.

6 Conclusions and Future Work

The main contributions of this paper are centered on the design and achievements of the experiment data repository in SCALEA which is a performance analysis tool for OpenMP/MPI and mixed parallel programs. We have described a novel design of SCALEA's experiment data repository holding all relevant experiment information and demonstrated several achievements gained from the employment of the data repository. The data repository has increasingly supported the automation of the performance analysis and optimization process.

However, employing the data repository introduces extra overheads in comparison with other non-employing-data-repository tools; the extra overheads occur in filtering and storing raw data to and retrieving data from the database. In the current implementation, we observed the bottleneck in accessing the data repository with large data volume. We are going to enhance our access methods and database structure to solve this problem. In addition, we intend to work on the issues discussed in Section 4.

References

1. T. Fahringer, M. Gerndt, Bernd Mohr, Felix Wolf, G. Riley, and J. Träff. Knowledge Specification for Automatic Performance Analysis, Revised Version. APART Technical Report, Workpackage 2, Identification and Formalization of Knowledge, Technical Report http://www.kfa-juelich.de/apart/result.html, August 2001.
2. T. Fahringer and C. Seragiotto. Automatic search for performance problems in parallel and distributed programs by using multi-experiment analysis. In *International Conference On High Performance Computing (HiPC 2002)*, Bangalore, India, December 2002. Springer Verlag.
3. Pallas GmbH. VAMPIR: Visualization and Analysis of MPI Programs. http://www.pallas.com/e/products/vampir/index.htm.
4. Karen L. Karavanic and Barton P. Miller. Experiment Management Support for Performance Tuning. In *Proceedings of Supercomputing'97 (CD-ROM)*, San Jose, CA, November 1997. ACM SIGARCH and IEEE.
5. Allen Malony and Sameer Shende. Performance technology for complex parallel and distributed systems. In *In G. Kotsis and P. Kacsuk (Eds.), Third International Austrian/Hungarian Workshop on Distributed and Parallel Systems (DAPSYS 2000)*, pages 37–46. Kluwer Academic Publishers, Sept. 2000.
6. Insung Park, Michael Voss, Brian Armstrong, and Rudolf Eigenmann. Parallel programming and performance evaluation with the URSA tool family. *International Journal of Parallel Programming*, 26(5):541–??, ???? 1998.
7. S. Pllana and T. Fahringer. UML Based Modeling of Performance Oriented Parallel and Distributed Applications. In *Proceedings of the 2002 Winter Simulation Conference*, San Diego, California, USA, December 2002. IEEE.
8. PostgreSQL 7.1.2. http://www.postgresql.org/docs/.
9. V. Taylor, X. Wu, J. Geisler, X. Li, Z. Lan, R. Stevens, M. Hereld, and Ivan R.Judson. Prophesy:An Infrastructure for Analyzing and Modeling the Performance of Parallel and Distributed Applications. In *Proc.of the Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC's 2000)*, Pittsburgh, August 2000. IEEE Computer Society Press.
10. Paradyn Parallel Performance Tools. http://www.cs.wisc.edu/paradyn/.
11. Hong-Linh Truong and Thomas Fahringer. SCALEA: A Performance Analysis Tool for Distributed and Parallel Program. In *8th International Europar Conference(EuroPar 2002)*, Lecture Notes in Computer Science, Paderborn, Germany, August 2002. Springer-Verlag.
12. Hong-Linh Truong, Thomas Fahringer, Georg Madsen, Allen D. Malony, Hans Moritsch, and Sameer Shende. On Using SCALEA for Performance Analysis of Distributed and Parallel Programs. In *Proceeding of the 9th IEEE/ACM High-Performance Networking and Computing Conference (SC'2001)*, Denver, USA, November 2001.
13. Felix Wolf and Bernd Mohr. Automatic Performance Analysis of Hybrid MPI/OpenMP Applications. In *Proceedings of the Eleventh Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP-11)*, pages 13–22. IEEE Computer Society Press, February 2003.
14. Omer Zaki, Ewing Lusk, William Gropp, and Deborah Swider. Toward scalable performance visualization with Jumpshot. *The International Journal of High Performance Computing Applications*, 13(3):277–288, Fall 1999.