BioOpera: Cluster-aware Computing

Win Bausch, Cesare Pautasso, Reto Schaeppi, Gustavo Alonso
Dept. of Computer Science
Swiss Federal Institute of Technology (ETHZ)
ETH Zentrum, 8092 Zürich, Switzerland
{bausch,pautasso,sreto,alonso}@inf.ethz.ch

Abstract

In this paper we present BioOpera, an extensible process support system for cluster-aware computing. It features an intuitive way to specify computations, as well as improved support for running them over a cluster, providing monitoring, persistence, fault tolerance and interaction capabilities without sacrificing efficiency and scalability.

1. Introduction

The increasing pervasiveness of clusters, combined with the sheer amount of data to be analyzed has made clusters a key tool in many scientific disciplines. One successful approach to programming in clusters is to use specialized problem solving environments. Such environments typically contain a library of ready-made components that can be combined, often through a GUI, into complex processes. These processes are themselves compiled into executable programs that automatically incorporate all necessary code for running the process over a cluster. This way, many of the complex details related to scheduling, resource allocation, and parallelism are hidden from the user. Although such platforms are not adequate for all types of cluster programming, they are very useful in a wide range of applications and constitute a much better option than scripting languages. The advantage of these platforms is that they provide a high level representation of complex computations. One that is much easier to understand and, therefore, to maintain and evolve than scripting languages. The basic principle they follow is an old idea: concentrate on high level composition rather than on low level programming [22]. This is achieved by pre-defining components that accomplish basic tasks and then providing a development environment supporting their composition into more complex entities. Examples of systems which follow very closely the idea of composition as the main programming primitive are TENT [10], Image 2000 [16], DISCWorld [11] or SECANT Technologies' Computing Farm Software [12].

In this paper we present the BioOpera system [7]. BioOpera takes the programming principle of composition beyond the simple graphical combination of pre-defined library components. In BioOpera, computations are represented as *processes*. A process is executed through an interpreter and its execution is combined with an *awareness model* similar to those developed for workflow management systems [5]. The awareness model centralizes and coordinates many different aspects of the execution, thereby facilitating the monitoring, interaction, and management of long-lived, complex computations over clusters.

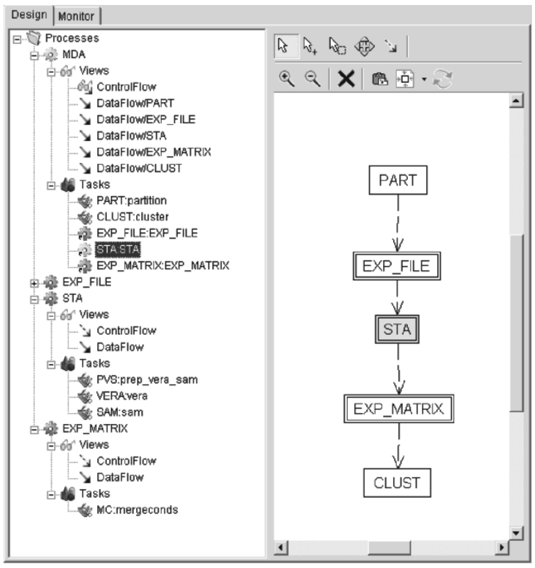
From the user's point of view, BioOpera acts as a high-level development and run-time environment for clusters and grid computing that provides an intuitive way to: (1) specify a parallel computation through composition by defining processes, (2) transparently schedule and coordinate the execution of processes over one or more clusters, (3) guarantee recoverability of all processes at all times, even after total failures, (4) provide a comprehensive environment for monitoring the execution of the processes and the state of the cluster(s), and (5) allow users to manage the computations by providing several interfaces (including a web based one) for, e.g., stopping, suspending, resuming, and canceling the execution. BioOpera is being actively developed at the Information and Communication Systems Research Group at the Department of Computer Science of ETH Zürich, and has already been successfully used to develop and manage long-lived computations on a set of clusters [3].

The rest of this paper is organized as follows. In Section 2 we present the programming model of BioOpera. Section 3 briefly sketches BioOpera's architecture. Section 4 describes the runtime services provided by BioOpera, focusing on management-related aspects. In Section 5 we show some performance measurements. Finally, Section 6 discusses Related Work and in Section 7 we conclude the paper.

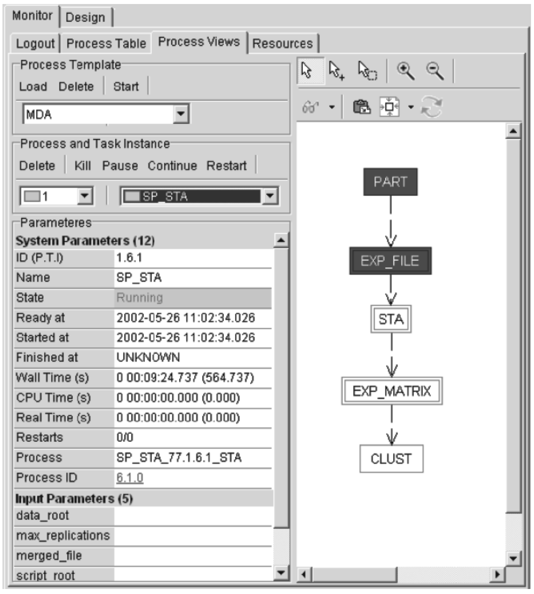
2. BioOpera from the user's point of view

Before discussing the architecture of the system, we describe how cluster based computations are developed using BioOpera. With this, we both motivate the architecture and introduce the system without getting into low level details.

For the user, a BioOpera computation is a workflow process constructed out of three elements: the computational process itself, clusters, and programs. In what follows, we describe how to define each one of these elements using BioOpera.



(a) Design tool



System Parameters (12)	
ID (P.T.I)	1.6.1
Name	SP_STA
State	Running
Ready at	2002-05-26 11:02:34.026
Started at	2002-05-26 11:02:34.026
Finished at	UNKNOWN
Wall Time (s)	0 00:09:24.737 (564.737)
CPU Time (s)	0 00:00:00.000 (0.000)
Real Time (s)	0 00:00:00.000 (0.000)
Restarts	0/0
Process	SP_STA_77.1.6.1_STA
Process ID	6.1.0
Input Parameters (5)	
data_root	
max_replications	
merged_file	
script_root	

(b) Monitoring tool

Figure 1. Process designer and monitor

2.1. Cluster computations as processes

BioOpera processes are suitable for expressing two basic types of cluster based computations. First, those constructed by combining different, possibly heterogeneous programs that constitute the successive steps of the computation. Typical examples of such computations are bioinformatic computations [9], image processing [16], or earth science models [4]. Second, those involving few, or even only one single computational step, but that need to be repeated for a range of parameter values. Examples of this second type of computations are repeated simulation runs, performed to obtain statistical significance, or exploratory analyses over wide parameter spaces [1].

Such computations are described in BioOpera as processes. A process consists of a set of *tasks* plus control and data flow dependencies among those tasks. Tasks can either be *activities*, involving the execution of an external program on one of the cluster nodes, or *sub-processes*, which are calls to other processes. Sub-processes are used to support modularity and code encapsulation. Both activities and sub-processes can be pre-defined and be made part of a library. BioOpera supports the definition of multiple such libraries that may be used by selecting a task within the library and dragging it into the process design tool.

A process' data flow needs to be explicitly specified by the developer. Control flow dependencies can be implicit or explicit. Explicit are again those defined by the developer. Implicit are those derived from the data flow: a task consuming data can only be started after all tasks producing the required data have successfully completed their execution.

When executing a process, BioOpera automatically analyzes the control flow dependencies and concurrently schedules all tasks that are found to be independent. If enough computing resources are available, these tasks will be executed in parallel. The developer has several possibilities for specifying parallel tasks. First, through control flow dependencies that evaluate to true at the same time (e.g., start all these tasks when task i finishes). Second, through specialized data flow connectors that dynamically unfold the tasks into multiple concurrent tasks as specified by the input parameters (equivalent to saying: execute this task n times over this set of n parameter values).

2.2. Configuring the cluster

Each node in the cluster must be registered with BioOpera before it can be used in a computation. Registration simply requires entering the corresponding host name or IP address. BioOpera automatically identifies the hardware and software settings of the node (number of CPUs, available memory and swap space, operating system). For large clusters, and to avoid having to register nodes one by one, entire ranges of IP addresses can be registered in a single operation. BioOpera also supports the definition of sub-clusters and arbitrary groups of nodes, including nested and overlapping ones. They are used for scheduling, access control, resource distribution (e.g., user A runs on one sub-cluster, user B on a different sub-cluster) and for describing the machines suitable for running a given activity.

The cluster configuration may be changed dynamically. Nodes may be added or removed at any time and BioOpera will adapt the running computations accordingly.

2.3. Activities and applications

Each activity in a process corresponds to an external application (a program or a system) that needs to be invoked. Before an activity can be mapped to an application, the application must be registered with BioOpera. Scientists can take already registered programs and use them as building blocks for processes, as well as register new software components with the BioOpera program library. This operation involves specifying the interface (input and output parameters) of the program, how to run it, and the range of nodes where it can be invoked. This information is also stored in a database and can be dynamically changed. BioOpera uses dynamic binding between activities and applications. Thus, it is possible to change, e.g., where an application must run while the process is executing (as long as the application has not been invoked already). Dynamic binding has significant advantages in terms of fault tolerance and software maintenance.

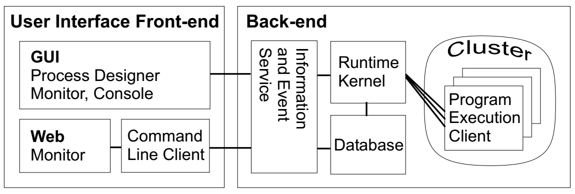
Conceptually, BioOpera is not tied to a specific type of program. Currently it can run applications running on Linux, Solaris, Windows or MacOS, call middleware objects through RPC or CORBA protocols, as well as perform SOAP Web Services invocations [20]. The system supports extending these basic application types with new ones.

Upon starting an application, BioOpera may send data to it. When the program completes, BioOpera may retrieve its results. For instance, to pass data to a UNIX application, BioOpera uses the application's command line arguments and may also write data directly to the standard input of the application. To retrieve any output that might be produced, BioOpera collects the data written to the application's standard output and standard error.

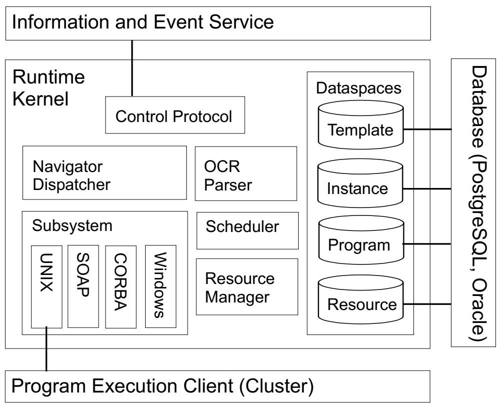
2.4. Development interface

Defining these three elements, processes, cluster configuration, and external applications, is done through the development interface of BioOpera (Figure 1.a). Processes are specified using a graphical tool that incorporates visual programming constructs. To implement a new process, the developer needs to define its input and output interface, as well as its content: the tasks and their dependencies. To insert a task into the process, the developer may select a program (or process) from the library, and drag it to the new process, thus inserting a new activity (or sub-process). A box for this new task is added in the control flow graph and boxes for the task and its input and output parameters are created in the data flow graph. The developer can then link the new task to other tasks in the process by drawing connections between them (control flow) or their parameters (data flow). The programming environment performs integrity checks to ensure the consistency of the control and data flow graphs drawn by the developer.

As soon as all the tasks and all the connections between them have been programmed, the process can immediately be executed, and interactively tested and debugged. Once completed and stable, it can be made available to be used as sub-process or to be invoked from the web interface.



(a) System components



(b) Runtime Kernel architecture

Figure 2. Architecture of BioOpera

3. System architecture

The BioOpera system is designed as an open multi-tier system. The kernel is extensible and provides basic persistent process enactment services. Care has been taken to ensure the flexibility of the architecture in order to support many different types of user interfaces, multiple program execution mechanisms and various database management systems.

3.1. System components

Functionally, BioOpera is divided into two layers: the User Interface Front-end and the Back-end (Figure 2.a). However each of those layers can be distributed in many different ways.

The Front-end is devoted to interacting with the user and contains the following tools: a *web based process monitor*, a *visual process development and monitoring environment*, and a *system administration console*. Additionally, a low-level *command line tool* is provided for building ad hoc extensions and integrating BioOpera with other software.

The Back-end includes: the *Runtime Kernel*, the *Database*, the *Program Execution Client* and the *Information and Event Service*. The Runtime Kernel has been designed as an generic and extensible kernel (Figure 2.b). It is in charge of process execution (scheduling and dispatching) and resource management. It is built on top of a database, used for making the process execution persistent and for storing resource and configuration information. Various execution subsystems can be plugged in, in order to support execution of different application types, e.g., UNIX or Windows applications, RPC calls or CORBA object method invocations and SOAP Web services. The database layer is divided into four *dataspaces*: template, instance, program, and resources, each of them dedicated to a different type of system data. Templates contain the structure of the processes. For each running instance of a process, the instance space contains a copy of the corresponding template and is used to persistently track the execution of a process. The program and resource dataspaces contain the information about applications and the cluster configuration. Each one of this dataspaces can be distributed across different nodes for performance purposes. The Program Execution Client (PEC) acts as a daemon in each node of the cluster. A copy of the PEC must be running on a node in order to allow BioOpera to schedule and execute jobs on that node. The PEC periodically reports the load and other state information of the node back to the resource manager. Finally, the Information and Event service bridges the user interface front-end with the rest of the system using an open SOAP-like API protocol.

3.2. Deployment scenarios

Each one of the components mentioned can be distributed to a different node. In addition, the database and the kernel can be replicated across several nodes for both fault-tolerance and performance purposes. A typical configuration involves one Program Execution Client running on each node of the cluster, one Runtime Kernel and Information Service running on the same or a different node, and a database server residing in yet another node. Multiple users may log on to the system through a web server or using the visual development environment as a remote client of the Information and Event Service. It is also possible to perform process invocations from one BioOpera kernel to another. This allows building complex hierarchical structures and grids consisting of different clusters. Interaction between the different BioOpera kernels can be either peer-to-peer (any server can invoke processes or sub-processes in another server) or master-slave (a master server controls the computation and uses the other servers to execute sub-processes as needed).

3.3. Executing a process

When starting a process, BioOpera creates a process *instance* which encapsulates all the information associated with the specific run. The instance is created from the corresponding template. Templates are created when a developer checks-in a finished process. Upon instance creation, BioOpera prompts the user for the values of the input parameters. At this time it is also possible to restrict the set of nodes used for execution (by default, BioOpera uses all eligible nodes).

To determine the set of tasks to execute, the BioOpera runtime engine analyzes the process control flow. This procedure is called *navigation*. BioOpera uses a main memory representation of the instance to perform navigation. This main memory representation is synchronized with a database representation used to record the state information associated to the instance. Multiple threads running in parallel perform navigation for different process instances. During navigation, the runtime engine accumulates information that needs to be made persistent. At the end of each navigation step, this information is stored into the database. This mode of operation minimizes the duration of individual transactions, thus avoiding unnecessary contention at the database level. Transactions are used to keep the information concerning process instances in a consistent state.

3.4. Awareness model

The awareness model incorporates runtime information and meta-data related to the processes as well as data pertaining to the environment of the cluster in which the computation takes place. This information includes the states of tasks and processes, profiling measurements, and execution logs, as well as the load and availability of each node. All together, this allows BioOpera to dynamically react to changes in the computing environment and provide the user with a very complete view of the computation.

The awareness model is implemented using a relational database management system. We have a fully functional PostgreSQL [15], as well as a partial ORACLE [17] implementation. Using a database management system has the advantage that information can be well organized and atomically updated, which proves very helpful in keeping the different views on awareness data consistent.

3.5. State of the computation

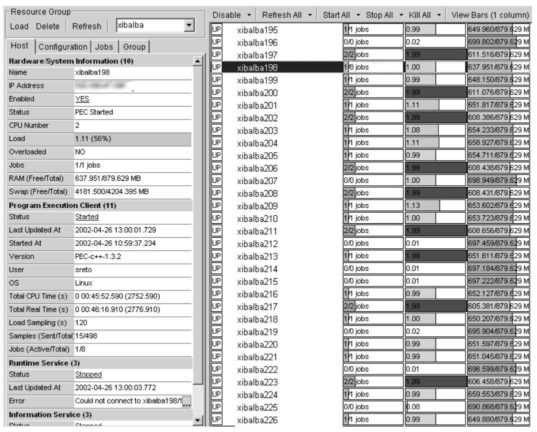
During execution, BioOpera keeps track of the current state of the computation in order to both transparently recover the processes from system failures as well as to react to various events that may occur during the execution.

While executing a process, BioOpera automatically profiles its execution, gathering performance measurements for all tasks, such as the CPU time (amount of processor work), the REAL time (accumulated execution time of all tasks) and the WALL time (duration of the computation). Part of this information is collected on the cluster nodes, and part of it is generated by timestamping state transitions of each task. This information may be used to identify bottlenecks in the structure of the process.

Another component of the awareness model is the data generated throughout the computation. This is used by BioOpera to trace the origin of results by mapping the data produced by the individual tasks to the data flow of a process.

3.6. State of the cluster

Finally, BioOpera gathers information regarding the CPU load in each cluster node, node availability, node failure, node capacity, the number of jobs being concurrently processed and other relevant information related to the state of the computing environment. This information is then



Hardware/System Information (10)	
Name	xibalba198
IP Address	[redacted]
Enabled	YES
Status	PEC Started
CPU Number	2
Load	1.11 (56%)
Overloaded	NO
Jobs	1/1 jobs
RAM (Free/Total)	637.951/879.629 MB
Swap (Free/Total)	4181.500/4204.395 MB
Program Execution Client (11)	
Status	Started
Last Updated At	2002-04-26 13:00:01.729
Started At	2002-04-26 10:59:37.234
Version	PEC-c++-1.3.2
User	sreto
OS	Linux
Total CPU Time (s)	0 00:45:52.590 (2752.590)
Total Real Time (s)	0 00:46:16.910 (2776.910)
Load Sampling (s)	120
Samples (Sent/Total)	15/498
Jobs (Active/Total)	1/8
Runtime Service (3)	
Status	Stopped
Last Updated At	2002-04-26 13:00:03.772
Error	Could not connect to xibalba198/1
Information Service (3)	

UP	xibalba195	1/1 jobs	0.99	649.960/879.629 M
UP	xibalba196	0/0 jobs	0.02	699.802/879.629 M
UP	xibalba197	2/2 jobs	1.99	611.516/879.629 M
UP	xibalba198	1/8 jobs	1.00	637.951/879.629 M
UP	xibalba199	1/1 jobs	0.99	648.150/879.629 M
UP	xibalba200	2/2 jobs	1.99	611.076/879.629 M
UP	xibalba201	1/1 jobs	1.11	651.817/879.629 M
UP	xibalba202	2/2 jobs	1.99	608.386/879.629 M
UP	xibalba203	1/1 jobs	1.08	654.233/879.629 M
UP	xibalba204	1/1 jobs	1.11	658.927/879.629 M
UP	xibalba205	1/1 jobs	0.99	654.711/879.629 M
UP	xibalba206	2/2 jobs	1.99	608.438/879.629 M
UP	xibalba207	0/0 jobs	1.00	698.949/879.629 M
UP	xibalba208	2/2 jobs	1.99	608.431/879.629 M
UP	xibalba209	1/1 jobs	1.13	653.602/879.629 M
UP	xibalba210	1/1 jobs	1.00	653.723/879.629 M
UP	xibalba211	2/2 jobs	1.99	608.656/879.629 M
UP	xibalba212	0/0 jobs	0.01	697.459/879.629 M
UP	xibalba213	1/1 jobs	1.99	651.611/879.629 M
UP	xibalba214	0/0 jobs	0.01	697.184/879.629 M
UP	xibalba215	0/0 jobs	0.01	697.222/879.629 M
UP	xibalba216	1/1 jobs	0.99	652.127/879.629 M
UP	xibalba217	2/2 jobs	1.99	605.381/879.629 M
UP	xibalba218	1/1 jobs	1.00	650.207/879.629 M
UP	xibalba219	0/0 jobs	0.02	695.904/879.629 M
UP	xibalba220	1/1 jobs	0.99	651.597/879.629 M
UP	xibalba221	1/1 jobs	0.99	651.045/879.629 M
UP	xibalba222	0/0 jobs	0.01	696.599/879.629 M
UP	xibalba223	2/2 jobs	1.99	606.458/879.629 M
UP	xibalba224	1/1 jobs	0.99	659.553/879.629 M
UP	xibalba225	0/0 jobs	0.08	690.868/879.629 M
UP	xibalba226	1/1 jobs	0.99	649.880/879.629 M

Figure 3. Cluster monitoring tool

used to balance the load between the different nodes, to schedule the computation according to machine usage and availability, and to resume the execution of the computation smoothly when failures occur.

4. BioOpera runtime services

In BioOpera, the management of the cluster and the computations running on the cluster is based on the awareness model described above. In order to cope with the different aspects of cluster management, BioOpera provides a set of runtime services that allows it to deal with each of these aspects individually. A first and well-known concern is failure masking: low-level failures, for instance failing nodes, need to be hidden from the user, in this case by rescheduling the jobs that were lost. A second issue is the monitoring of a computation: especially when running complex, long-lived computations, users will want to be informed about the progress of a computation, or about failures that cannot be handled automatically by the system. Third, they will want to be able to intervene in the computation as they see fit, e.g., by aborting and restarting parts of it. As fourth concern, system administrators need to be able to assess the utilization of the cluster or the effect of node outages on active computations. Last, data resulting from a computation needs to be analyzed and interpreted, which may involve looking at how the data was produced throughout the computation. Each of the following sections is dedicated to one of these five concerns. Further runtime services can easily be implemented since all the necessary information is available on the system's database. In most cases, this involves only defining new queries over the accumulated data and developing the corresponding interface.

4.1. Automatic failure handling

BioOpera is designed to provide a dependable cluster computing environment on top of an unreliable, error-prone hardware/software infrastructure. To this end, several types of failures are transparently handled: cluster node hardware failures, internal system failures, and program failures.

In the first case, BioOpera periodically polls each node in the cluster querying it for the status of its running jobs. Both if a node doesn't respond after a certain user defined time, and if the node replies that the requested jobs are unknown, BioOpera assumes the node has failed (in the second case, it has probably already been rebooted) and will transparently reschedule the lost jobs on a different node of the cluster.

Internal system failures encompass all situations that eventually require restarting a BioOpera system component. BioOpera has been designed such as to offer persistent process execution, in the sense that running process instances will not be affected by failures in the BioOpera system components. Information relevant for recovery is updated throughout the execution of a process instance and is used to recover its state after restart. Relevant information for each task includes its state and the address of the cluster node it is running on. To increase the level of reliability, the underlying database can be replicated or a hot-standby configuration used.

Concerning program failures, BioOpera detects whether a program has successfully been executed based on its exit code. If a program returns a zero exit code, BioOpera sets the corresponding task to finished, in all other cases the task will fail. The developer may specify additional behavior, such as automatically restarting a task up to a certain number of times before setting it to failed, or executing some other tasks responsible for handling the exception.

In any case, when a task fails, the user may manually restart it, possibly after fixing the problems with the application that caused the failure.

4.2. Process monitoring

Once a process instance has been started, the user may follow its progress and check intermediate results as BioOpera schedules and executes the various tasks on the cluster. At all times, BioOpera presents the user with a clear, high-level view over the state of the computation running on the cluster (Figure 1.b).

As the computation progresses, the user may watch the various tasks changing color and, for instance, easily differentiate failures (e.g. colored in red) from successful program executions (e.g. colored in blue). This simple but intuitive approach is very helpful when monitoring hundreds of concurrent processes. BioOpera also keeps track of the data that is produced by the tasks. This way, the user may read the value of the data flow parameters and, in the case of UNIX applications, the content of the standard output and standard error produced by a program after and even during its execution. While checking the state of a task helps to detect software failures, analyzing its output ultimately helps to understand and fix the error that caused them. Being able to do this from a centralized point is a significant advantage when the computation is complex and distributed over many nodes.

Furthermore, together with the state information, BioOpera records the host's name a task is running on. This enables BioOpera to periodically check whether running jobs are still healthy. Additionally, it allows the user to easily switch between a process and a cluster centered monitoring view (Figures 1.b and 3), going back and forth between watching the progress of a process and overseeing the state of the entire cluster over which the process runs. The user may query a node for performance related information, such as processor load, free memory and swap space, as well as ask for the list of tasks running on it.

4.3. Interacting with a process

The fact that BioOpera keeps track of all dynamic state information associated with a running computation opens up new possibilities to interact with it. BioOpera provides a set of predefined signals that can be sent either to individual tasks or to the process as a whole. Currently supported signals are *kill*, *suspend*, *resume* and *restart*.

4.4. System administration

Given the functionality described above, it is possible to plan ahead concerning the operation of the cluster. This is a key feature when running processes that may last months and, therefore, are likely to encounter many different situations (in addition to failures): the need to upgrade software and hardware, the replacement of nodes in the cluster, changes in storage devices, and so forth.

In this regard, BioOpera has several advantages. Since the computation is outlined in the process, it is possible to determine which process instances would be affected when a given node is taken off-line. This gives system administrators a very powerful tool to perform upgrades and changes to the system as the computation proceeds while minimizing the impact of the outages. Thanks to the dynamic scheduling and load balancing mechanisms, as well as to the ability to group cluster nodes in named units, BioOpera is capable of working with a cluster that shrinks or grows in size dynamically. It is even possible to replace tasks initially intended to run in a given node with alternative tasks running on a different node (even on a different OS).

4.5. Data analysis

BioOpera keeps information about processes that are currently running on the cluster, but also about processes that have completed their execution, either successfully or with failure. The user may search this information to find out more about the history of a particular run, for instance to debug software failures, to determine how certain results have been produced, to compute the resource utilization of the cluster over a certain time period and for a certain set of processes, or to list all hardware failures in the cluster. Additionally, old processes may be restarted using the same input parameters, or the user may repeat some runs changing the values of some input parameters.

The fact that all the necessary information is well organized and stored in a relational database opens up the opportunity to create very sophisticated automatic tools for system administration, on-line computation analysis and optimization that go well beyond anything available today. Any search that can be expressed with an SQL statement is supported. For instance, it is possible to query the database for a list of all failures on a specific cluster which occurred within a certain time interval, concerning the processes run by a particular user.

5. BioOpera in practice

To demonstrate how BioOpera makes managing a computation over a cluster easy without sacrificing performance we discuss several complex computations performed under BioOpera.

First of all we present an experiment which involved running computations lasting for over a month. This shows how the system reliably sustained the computation and transparently dealt with various failures in the cluster as well as with changes in its configuration. Second, we show a scalability experiment, in which BioOpera runs the same process over an increasing number of cluster nodes (from 1 to 60). Third, we add a throughput experiment to measure the average process startup time and the size of the workload, measured in running activities per unit of time, the current BioOpera prototype can handle.

5.1. Experimental setup

The experiments have been performed using different cluster of PCs and UNIX workstations linked by an ordinary Ethernet 100Mbit network. The following table summarizes their main hardware and software characteristics.

	Nodes	CPU (Mhz)	RAM (MB)	OS
L.	16	P-III (500)	512	LINUX v2.2.12
K.	5	UltraSPARC (269)	192	SUNOS v5.6
I.	8	P-III (600)	512	LINUX v2.2.14
X.	60	P-III (1000)	1024	LINUX v2.4.17

All the nodes in the Linux clusters have 2 CPUs. During the first experiment the cluster was shared with other users, while in the others the cluster was managed exclusively by BioOpera.

5.2. Stability and reliability

The first experiments involved a process computing the self comparison (or *All vs. All*) of the protein sequence database SwissProt version 38 [8] (see [3] for details). Before switching to BioOpera, this process took several months to compute, mainly due to the overhead of manually managing the computation. For earlier versions of SwissProt (with less data) the computation required about 400 days of CPU time and lasted two to three months in real time.

To use BioOpera, a process describing the computation was built and then executed using clusters *I*, *K*, *L*. As presented in [3], BioOpera made it easier to deal with changing conditions in the cluster, ranging from user-driven interruptions (e.g. other users requesting exclusive access to the cluster) to hardware problems (e.g. disk space shortage).

5.3. Scalability

To test the scalability of processes run under BioOpera, we have been using a process analyzing data from a gene expression profiling experiment based on cDNA microarrays. The underlying algorithms, as well as the overall procedure are described in [14]. The data used for the test run was downloaded from the Stanford Microarray Database [18] and has already been analyzed as part of [2]. The most time consuming step in the analysis (∼96% of the overall CPU time) is the computation of the likelihood that a specific gene is differentially-expressed [13]. This likelihood needs to be calculated for each experimental condition, each of which may be analyzed independently of the other. Our data set consisted of 66 different conditions. The degree of parallelism between tasks is left to BioOpera to decide.

Figure 4 shows the results of the test run. During the run, cluster *X* was exclusively used by BioOpera. The left vertical axis shows the WALL time (time from beginning to end of the computation), the right vertical axis the CPU time (time spent computing) and REAL time (time spent in each node including computing and I/O waits). The horizontal axis indicates the number of nodes, each of them with 2 CPUs. The results prove that the process scales well up to 35 available nodes. Beyond 35 nodes, however, there is no improvement to be observed. This is due to the fact that BioOpera always schedules two concurrent tasks on a single cluster node so that, for this process, beyond 35 nodes there is no more parallelism to be exploited. Overall, the experiment demonstrates that BioOpera can be used to parallelize computations and obtain performance gains without having to become familiar with sophisticated programming techniques.

5.4. Throughput

To illustrate the throughput (activities per unit of time) that can be achieved using BioOpera and the implications of persistence, we have been using a synthetic process involving two independent control flow threads, each thread consisting of 150 sequentially ordered activities. Each activity computes between 15 and 30 seconds. We have been running 200 concurrent instances of this process on 30 nodes of the *X* cluster. To be able to test the raw throughput of the system, BioOpera's scheduler has been configured to ignore the node's workload in placement decisions. All instances were run using the same priority.

Figure 5 shows the results of this test run. The horizontal axis represents time, the vertical axis counts the number of activities. The overall wall time of the run was 1 hour, 9 minutes and 10 seconds. For time t, the graph displays the number of activities that have been submitted to the scheduler to be run (fat line), and number of activities that have actually been scheduled and are running in the cluster (thin line).

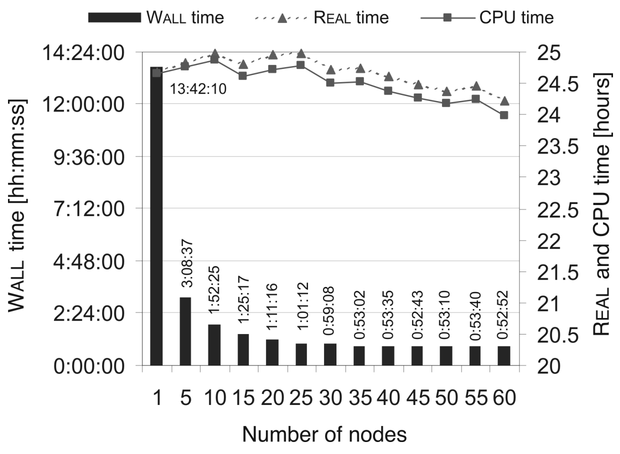


Figure 4. Scalability measurements

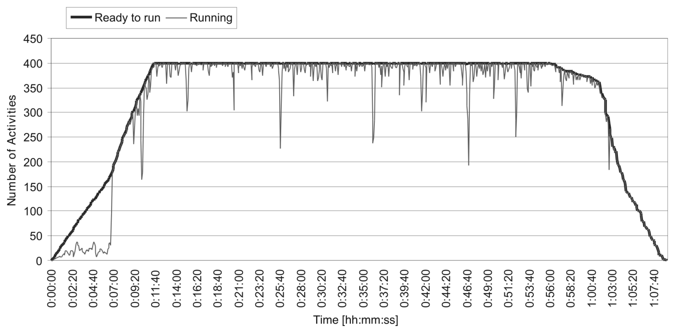


Figure 5. Throughput measurements

The only noticeable problem when dealing with such a high load is that the time it takes to start all 200 instances is quite long (about 11.5 minutes). Per instance, however, this only amounts to 3.45 seconds of instantiation overhead. The overhead arises from the need to create an image of the process in main memory and in the database. This procedure, however, can and will be optimized in future versions of the system. Keeping that in mind, and in view of the results of the test, we are confident that BioOpera will introduce only a minimal overhead when dealing with concurrent processes. Note that, the instantiation problem aside, BioOpera had no difficulties in using all available resources to schedule tasks as they became ready for execution.

6. Related work

Conventional workflow management systems (WFMS) are generally inadequate for modeling and running cluster computations [21]. However, many ideas borrowed from

WFMSs have been used to great advantage in cluster computing. TENT [10] uses the same programming model as workflow systems for parallelizing numerical simulations. It lacks, however, persistence and a sophisticated awareness model for cluster management. Image2000 [16] also uses a workflow approach for describing image processing but no attempt is made at exploiting parallelization. Its main application area is in astronomy although it can be tailored to other domains involving image processing. BioNavigator [9] uses the process concept to provide biologists with a web based research platform to perform in silico experiments. Research done as part of the TAMBIS [6] project also advocates a workflow-like notation to describe bioinformatic tasks [19].

All these tools focus on the idea of process as programming model, some exploit the process structure, e.g., to be able to provide the user with a detailed view of the data involved in a computation but none of them do it to the extent we propose in this paper.

7. Conclusion

BioOpera provides a powerful software infrastructure for rapidly building distributed applications and efficiently run them over a cluster of computers. BioOpera uses a simple but powerful visual language to facilitate composition based programming. BioOpera supports user monitoring and interaction with running processes, transparent failure handling (through rescheduling and restarting), automatic gathering of performance measurements and browsing through the history of past executions. Experiments have shown that our system achieves low overhead, good performance and scalability over a cluster of workstations. We are currently testing and tuning the system, and using it to develop several bioinformatic applications.

References

[1] D. Abramson, R. Sosic, J. Giddy, and B. Hall. Nimrod: A tool for performing parametrised simulations using distributed workstations. In *Proc. 4th IEEE Symposium on High Performance Distributed Computing*, pages 112 –121, Washington DC, USA, 1995.

[2] A. A. Alizadeh, M. B. Eisen, R. E. Davis, C. Ma, I. S. Lossos, A. Rosenwald, J. C. Boldrick, H. Sabet, T. Tran, X. Yu, J. I. Powell, L. Yang, G. E. Marti, T. Moore, J. Hudson, J., L. Lu, D. B. Lewis, R. Tibshirani, G. Sherlock, W. C. Chan, T. C. Greiner, D. D. Weisenburger, J. O. Armitage, R. Warnke, L. M. Staudt, and et al. Distinct types of diffuse large b-cell lymphoma identified by gene expression profiling. *Nature*, 403(6769):503–11., 2000.

[3] G. Alonso, W. Bausch, C. Pautasso, M. Hallett, and A. Kahn. Dependable Computing in Virtual Laboratories. In *Proc. of the 17th International Conference on Data Engineering (ICDE2001)*, pages 235–242, Heidelberg, Germany, 2001.

[4] G. Alonso and C. Hagen. Geo-Opera: Workflow Concepts for Spatial Processes. In *Proc. 5th Intl. Symposium on Spatial Databases (SSD'97)*, pages 238–258, Berlin, Germany, 1997.

[5] D. Baker, D. Georgakopoulos, H. Schuster, A. R. Cassandra, and A. Cichocki. Providing customized process and situation awareness in the collaboration management infrastructure. *CoopIS*, pages 79–91, 1999.

[6] P. G. Baker, A. Brass, S. Bechhofer, C. Goble, N. Paton, and R. Stevens. TAMBIS: Transparent access to multiple bioinformatics information sources. In J. Glasgow, T. Littlejohn, F. Major, R. Lathrop, D. Sankoff, and C. Sensen, editors, *6th Int. Conf. on Intelligent Systems for Molecular Biology*, pages 25–34, Montreal, Canada, 1998. AAAI Press, Menlo Park.

[7] BioOpera. Process support for bioinformatics. www.inf.ethz.ch/department/IS/iks/project_home_pages/bioopera/.

[8] G. Cannarozzi, M. Hallett, J. Norberg, and X. Zhou. A cross-comparison of a large gene dataset. *Bioinformatics*, 16:654–655, 2000.

[9] Entigen. *BioNavigator*. http://www.bionavigator.com.

[10] T. Fokert, H.-P. Kersken, A. Schreiber, M. Striezel, and K. Wolf. The distributed engineering framework TENT. In J. M. L. M. Palma, J. Dongarra, and V. Hernández, editors, *Vector and Parallel Processing - VECPAR 2000, 4th International Conference, Porto, Portugal, June 21-23, 2000, Selected Papers and Invited Talks*, volume 1981 of *Lecture Notes in Computer Science*, pages 38–46. Springer, 2001.

[11] K. Hawick, H. James, C. Patten, and F. Vaughan. Discworld: A distributed high performance computing environment. In P. Sloot, M. Bubak, and B. Hertzberger, editors, *Proc. of High Performance Computing and Networks (HPCN), Amsterdam, 1998*, volume 1401 of *Lecture Notes in Computer Science*, pages 598 – 606. Springer, 1998.

[12] S. Herb and J. Ponpeii. Linux cluster in life sciences demonstration. *LinuxWorld Conference and Expo, New York, USA*, 2001. Also available as IBM White Paper: www-4.ibm.com/solutions/lifesciences/pdf/LifeSciencesLinuxCluster.pdf.

[13] T. Ideker, V. Thorsson, A. F. Siegel, and L. E. Hood. Testing for differentially-expressed genes by maximum-likelihood analysis of microarray data. *Journal of Computational Biology*, 7(6):805–17., 2000.

[14] Institute for Systems Biology. *The analysis pipeline at the Instiute for Systems Biology*. http://www.systemsbiology.org/ArrayProcess/index.html.

[15] B. Momijian. *PostgreSQL: Introduction and Concepts*. Addison Wesley, 2000. http://www.postgresql.org.

[16] NASA. *Image 2000*. http://icc.gsfc.nasa.gov/image2000/.

[17] ORACLE Corp. *ORACLE 8i*. http://www.oracle.com.

[18] Stanford University. *Stanford Microarray Database*. http://genome-www5.stanford.edu/MicroArray/SMD/.

[19] R. Stevens, C. Baker, P. Baker, and A. Brass. A classification of tasks in bioinformatics. *Bioinformatics*, 17(2):180–188, 2001.

[20] W3C. *Simple Object Access Protocol (SOAP) 1.1*, 2000. http://www.w3.org/TR/SOAP.

[21] M. Weske, G. Vossen, and C. Medeiros. Scientific workflow management: WASA architecture and applications. Technical Report 03/96-I, Universitat Munster, 1996. http://dbms.uni-muenster.de/staff/weske/papers/fb03-96.ps.gz.

[22] G. Wiederhold, P. Wegner, and S. Ceri. Towards megaprogramming: A paradigm for component-based programming. *Comm. ACM*, 35(11):89–99, 1992.