Automatic, multi-grained elasticity-provisioning for the Cloud

Translational Cancer Detection Pipeline over CELAR Application and Evaluation

Deliverable no.: D8.3
Date: 29-09-2015

CELAR is funded by the European Commission DG-INFSO
Seventh Framework Programme, Contract no.: 317790

Table of Contents

List of Figures

List of Abbreviations

NGS	Next Generation Sequencing
API	Application Programming Interface
DB	Database
GUI	Graphical User Interface
PaaS	Platform as a Service
UML	Unified Modeling Language
VM	Virtual Machine
WP	Work Package

Deliverable Title	**Translational Cancer Detection Pipeline over CELAR Application and Evaluation**
Deliverable no	**D8.3**
Filename	CELAR_D8.3.docx
Author(s)	Christopher Smowton, Wei Xing
Date	2015-09-29

Start of the project: 01-10-2013
Duration: 36 months
Project coordinator organization: ATHENA RESEARCH AND INNOVATION CENTER IN INFORMATION COMMUNICATION & KNOWLEDGE TECHNOLOGIES (ATHENA)

Due date of deliverable: 30-09-2015
Actual submission date: 30-09-2015

Dissemination Level

X	PU	Public
	PP	Restricted to other programme participants (including the Commission Services)
	RE	Restricted to a group specified by the consortium (including the Commission Services)
	CO	Confidential, only for members of the consortium (including the Commission Services)

Deliverable status version control

Version	Date	Author
1	2015-07-20	Christopher Smowton, Wei Xing
2	2015-09-21	Christopher Smowton, Wei Xing
3	2015-09-22	Christopher Smowton, Wei Xing
4	2015-09-28	Christopher Smowton, Wei Xing
5	2015-09-29	Christopher Smowton, Wei Xing
6	2015-12-08 [post-review]	Christopher Smowton, Wei Xing

Abstract

The aim of CELAR WP8 is to develop a cloud-based cancer detection application (SCAN) for translational cancer research. The key objective of the SCAN development is to manage computation and data elasticity in biological applications using the CELAR cloud platform. In this technical report, we present the final design of the SCAN biological analysis platform, describe its implementation and evaluate its performance, and specifically its ability to scale scientific computing, against three biological analysis pipelines that are representative of the best practices in biocomputing.

SCAN is a Resource Manager (RM) that coordinates the execution of end-users' biological analyses using a dynamically mutable pool of worker virtual machines. It improves over existing RMs with better support for the elastic cloud execution scenario, and by introducing a novel metric based on user happiness that helps to inform elastic cloud scaling decisions.

Keywords

Translational Cancer Research, Cancer Detection Pipeline, CELAR System Architecture, Automated Elasticity Provisioning, Resource Allocation, Application Management Platform, Elasticity Platform, Cloud Computing, Workflows

1 Introduction

In the field of biological computing, a diverse array of scientific programs and algorithms are used to analyze and model biological processes, from determining DNA mutations to modeling molecular structures and interactions.

SCAN is a platform for executing those analysis programs in an elastic cloud environment. It is a resource manager, keeping track of queues of user analysis requests and a pool of workers that can run them. It is particularly suited to the elastic cloud in that it exposes metrics indicating how well it is currently performing, and allows run-time addition, removal and reconfiguration of workers.

This kind of elastic cloud resource manager can benefit biological researchers by freeing them from having to manually allocate resources to their analyses, which often-non-expert users are apt to do poorly.

A system or person, called SCAN's Supervisor, is typically expected to monitor SCAN's performance metrics and therefore dynamically adjust the number and type of workers that are placed at SCAN's disposal. CELAR works with SCAN by playing this Supervisor role.

Biological computing tasks are, in practice, typically executed using a cluster resource manager, such as the classic Portable Batch System or a more modern system such as Apache YARN. SCAN aims to match and improve upon these systems' capabilities, and particularly exploit opportunities presented by the elastic cloud:

- Under most widely-deployed resource management schemes, users or administrators are responsible for determining the distribution of resources between jobs (for example, how many cores should a multi-threaded job request?). SCAN adaptively allocates resources, taking into account both the observed properties of the user's task and the current workload level SCAN is experiencing.

- SCAN dynamically profiles users' jobs, characterizing the relationship between their input data and running time, and the relationship between resource allocation and performance (for example, how does execution time scale with thread count?)

- SCAN maintains a distributed filesystem (DFS) and optimizes job assignment to maximize data locality. This is similar to the DFS used with modern cloud-oriented resource managers, such as Apache YARN, but is suitable for use with classical workloads that are more often run in cluster environments.

SCAN was designed at the Cancer Research UK Manchester Institute. Thus while SCAN is a general-purpose resource manager, suitable for any variety of scientific computing or indeed any field that needs job distribution across an elastically allocated worker pool, the authors' particular use case centers around biocomputing and more specifically cancer analysis. Cancer research requires large-scale computing because the instruments used in the field, such as DNA sequencers, often produce large datasets that require significant processing before useful information can be extracted. Performing these analyses quickly is also vital if they are to be made practical for use in day-to-day medicine, for example providing diagnosis from a tumor biopsy within an acceptable time frame. Cancer research also produces highly variable demand for computing resources due to short-term shifts in researchers' focus; thus it stands to benefit greatly from elastic scaling of the compute resources employed.

SCAN's design was previously sketched in deliverable D8.1 [**1**] and elaborated in D8.2 [**2**,**1**] along with details of its initial implementation. We also benchmarked the version of SCAN described in D8.2 ("SCANv1") [**3**] and studied the opportunities for dynamic resource allocation to SCAN jobs [**4**] since publishing D8.2. In this document, we describe extensions of SCANv1's design (producing SCANv2), describe the implementation of SCANv2, and evaluate SCANv2's functionality using three sample biological applications, picked to exhibit a range of different resource requirements typical of different sub-fields of biocomputing. Whilst SCANv1 was originally specified and designed to tackle cancer research problems, in SCANv2 we generalize to address biological bio-computing in general, which we argue has similar needs.

We describe SCANv2's design in section 2, give details of its implementation in section 3, characterize the biological applications we use to test it in section 4, evaluate SCANv2 in section 5, and finally conclude in section 6.

2 Design

SCAN is a resource manager for biological analysis execution in an elastic cloud environment. SCAN is agnostic of the specific kind of work the users wish to execute in the cloud. However, within the specific subfield of cancer research SCAN can be applied to manage common workflows such as DNA mutation and structural variation analysis, protein identification from mass spectrometry or chemical and physical simulation.

2.1 Definitions and Terms

Users are humans that have analysis programs they wish to execute in an elastic cloud environment.

The **Administrator** is a human that sets up SCAN at the start of the day.

The **Supervisor** is a human or automated system that continually adjusts the resources (virtual machines and their virtual hardware, including cores, memory and disks) provided to SCAN.

The **Infrastructure Provider** supplies those resources.

Jobs are the units of work Users submit to SCAN for execution. They consist of a shell script for execution and a *class*. They may be annotated with an estimated size.

Job classes specify which jobs may be regarded as similar, and whose estimated sizes are comparable, for the purpose of performance modeling. Comparing the estimated sizes of jobs of differing class is meaningless.

Composite tasks are workflows consisting of multiple jobs: for example, one might produce an intermediate result that another consumes.

In designing SCAN, we aim to provide a service that is easy for the User to interact with, in that it is easy for them to supply SCAN with enough information to run their jobs effectively. It should also be informative to the Supervisor, so that they can practically devise policy relating SCAN's performance metrics to the resources it should be given, and of course should run User workloads more effectively, and with less effort, than the average user coordinating their workload manually.

2.2 Actors

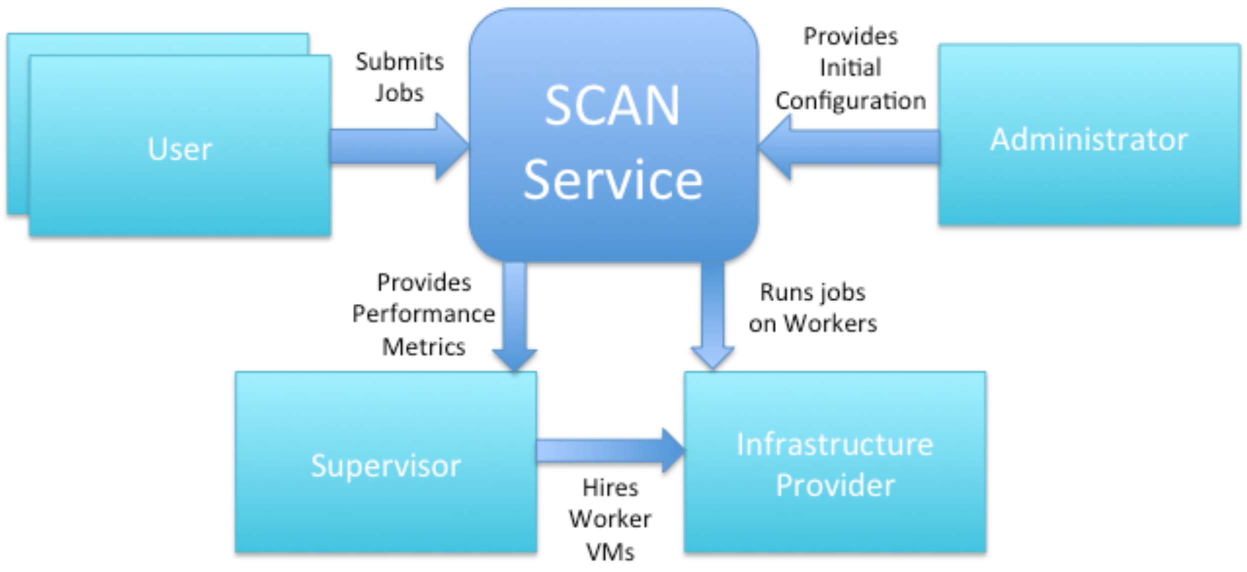


Figure 1: Diagram of Actors and relationships in the SCAN system

Figure 1 illustrates the key actors involved with SCAN: the Administrator who sets the SCAN service up to begin with, the Users who submit work for SCAN to execute, the Supervisor (which may be a human or another system, such as CELAR) who monitors SCAN's performance, and the Infrastructure Provider from which the supervisor acquires workers that the SCAN Service will use to execute Users' jobs.

2.2.1 Administrator

The administrator's role takes place entirely at initialization time, before any jobs are submitted or executed. They must provide initial parameters indicating the classes of task that users can submit, the target completion time for such tasks, and the relationship between input data size, degree of multithreading and completion time. These measures are defined in more detail in Section 2.4, and serve only as initial values, being replaced with observed values after sufficient runs that dynamic profiling data becomes meaningful.

The administrator must also commission a virtual or physical machine on which to run the SCAN Scheduler (its central coordinating component), reachable from both potential worker machines and potential users. They may use the Supervisor's virtual-machine commissioning mechanism to achieve this, as is standard practice when CELAR plays the Supervisor role.

They can also provide *templates* for users, which ease the process of job submission by providing shell scripts that can be parameterized, and *viewers*, which provide users with convenient means to observe files in the SCAN system without having to analyse them offline.

2.2.2 User

Users submit jobs to SCAN which they wish to be executed. They can be submitted as an arbitrary shell script, or by specifying the name of a template defined by the administrator along with zero or more parameters. They must also annotate each job with one of the classes defined by the administrator, indicating that the job is expected to share performance characteristics (response to data input size and multithreading) with other jobs having the same class, and should use the completion time targets associated with that class.

Users' jobs may make use of the SCAN file system, in which case the user should provide a list of required inputs and expected outputs when submitting the job (this is described in more detail in Section 2.6). These allow SCAN to optimize for and track data locality. Users may upload and download files to/from the SCAN file system, but can also have their jobs source data from or post data to an external resource, such as a cloud data service.

The SCAN Service does not take responsibility for dependency-ordering jobs; however, SCAN comes with a plugin for the GATK Queue job management utility, which provides this functionality client-side.

2.2.3 Supervisor

The Supervisor's role is to monitor SCAN, and, on the basis of its observed performance, provide it with virtual or physical machines to execute its workload. The service informs the supervisor of basic statistics, such as the queue length, the workload makeup in terms of classes, and the hardware utilization experienced by its existing workers (e.g. CPU and memory utilization). The supervisor should thus choose how many workers to provide to SCAN, and the hardware specification of each one. When workers are to be withdrawn from service, it should choose which one to take away.

It should also monitor the storage needs of the workers, which cooperate to implement the SCAN file system, and dynamically adjust the storage attached to each one to achieve balance.

The supervisor role can be played by a human (probably the same as the administrator), but for our purposes will always be fulfilled by the CELAR system.

2.2.3.1 *Distinguishing SCAN and Supervisor Roles*

SCAN and its Supervisor play distinct but closely related roles, which can be easily confused, so we will explicitly distinguish the two here. SCAN's role is similar to cluster- or cloud-based resource managers such as PBS (a cluster resource manager) or YARN (a cloud resource manager). Their goal is to dispatch users' jobs against a pool of workers. A resource manager's pool of workers is usually either static (as in a typical cluster scenario) or under the manager's direct control. SCAN, by contrast, manages a dynamically variable pool of workers. Workers' size and number are not chosen, nor are they hired by SCAN. They are dynamically allocated in different numbers, sizes by its external Supervisor, in this case the CELAR system. SCAN needs only to provide CELAR with operational statistics in order for scaling decisions to be reached. Hence, CELAR observes SCAN's performance and decides how many and what kind of workers SCAN should be supplied with, while SCAN controls how those workers are shared between its users.

2.2.4 Infrastructure Provider

The infrastructure provider is whatever service the supervisor employs to provide worker physical or virtual machines for use by the SCAN Service. It should be able to host GNU/Linux instances suitable for running users' required programs, as well as SCAN's required utilities, such as an SSH server. In this document's evaluation section we use KVM virtual machine instances provided through the ~Okeanos service, but other suitable technologies include lightweight container systems such as LXC, or even systems that do not implement container isolation such as a classical cluster scheduler.

In order for the Supervisor to scale the number and specification of workers provided to scan, the Provider should be capable of dynamically acquiring and releasing a variety of different kinds of virtual or physical machine. Ideally it should also be able to attach and detach storage devices from an active worker, thus enabling dynamic storage allocation within the SCAN file system.

2.3 Architecture

The SCAN Service implements four key functionalities: scheduling users' jobs to workers, modeling and predicting those jobs' performance, managing the SCAN filesystem, and exposing informative performance metrics to its Supervisor.

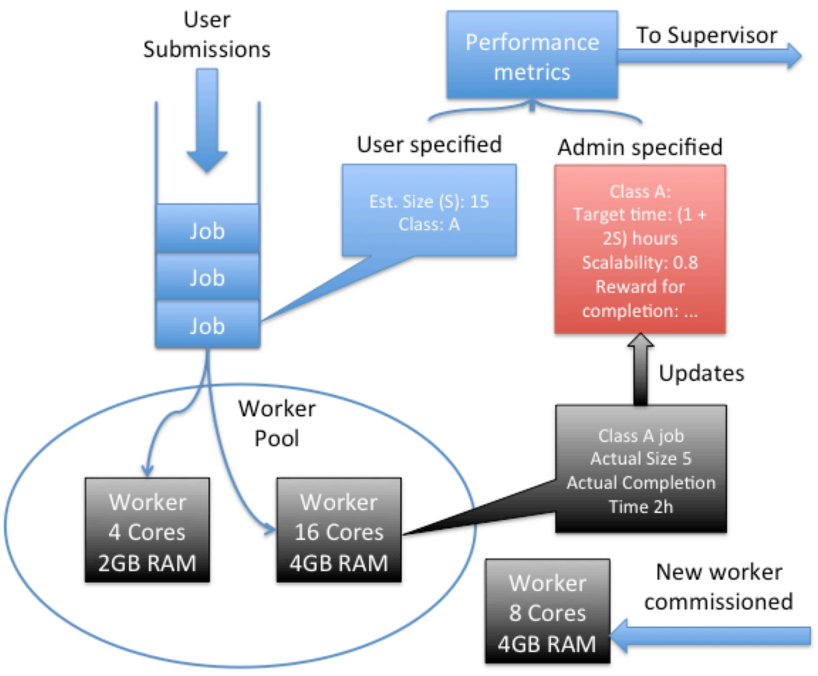


Figure 2: Core functionality of the SCAN Service

Figure 2 illustrates its core functionalities unrelated to storage management, which is explored in more detail later: accepting jobs submitted by users, along with annotations that help SCAN predict its likely duration, scheduling those jobs against workers, monitoring job execution to generate a superior model in the future, and providing performance metrics to its Supervisor. All depicted functionality is part of the SCAN Scheduler, except for the Workers themselves, which are separate components, and some elements of user job submission, which are executed client-side.

2.4 Scheduler

The SCAN Scheduler is the primary component of the Service. It manages the Worker pool and job queue and is responsible for computing most performance metrics that are exposed to the Supervisor.

2.4.1 Jobs and Reward

As mentioned previously, Users submit Jobs to the Scheduler as either arbitrary shell scripts, or by naming a template defined by the administrator and supplying parameters, which the scheduler substitutes into the template to produce a shell script for execution. For example, within the domain of cancer research, a job might run a step in a DNA mutation analysis pipeline.

The scheduler maintains a single FIFO queue of pending jobs, and attempts to run each job in turn on one of the workers in its pool.

The user supplies an estimated size for each job they submit, indicating in arbitrary units the amount of work that is expected relative to other jobs of the same class (for example, for many genome analysis tasks the size in bytes, or number of records, of the input file suffices as a job size estimate). The Scheduler selects the optimal degree of parallelism based on this estimate, parameters supplied by the administrator, and the current resources available in the worker pool.

The rules for selecting the appropriate degree of parallelism (number of cores assigned) are as follows:

- The user may specify an upper limit, which is never exceeded
- Never try to assign more cores than the worker with the most cores (occupied or not).
- For each possible number of cores up to whatever limit applies, find the core count that will yield the highest Reward (an arbitrary value approximating user happiness). SCAN calculates the expected Reward as follows:

The administrator provides (and the Scheduler successively refines) the job class' **expected degree of parallelism, P,** the **constant time overhead, c,** the **time per unit work, t,** the **target completion time, T_0** and the **reward per hour early/late, R_h.** For a job of **estimated size S** running with **n cores**, having **queued for Q hours** and with **expected waiting time W**, this yields the expected reward **R**:

$$R(S,n,W) = R_h(T_0 - (Q + W + (c + tS)\left((1-P) + \frac{P}{n}\right)))$$

This consists of the **expected single-threaded time** $c + tS$, corrected for the number of cores assuming that a proportion of its runtime P is perfectly scalable whilst the remaining $1 - P$ does not scale, assigned reward per hour it differs from the target completion time T_0.

The waiting time W depends on the current state of the worker pool: if a worker already has n cores free it is zero; otherwise it is set based on when enough currently-running jobs are expected to release enough worker cores, or when the Supervisor is expected to supply an extra worker, whichever is sooner.

The scheduler selects the degree of parallelism with the highest expected reward. This means that whilst with idle workers and non-zero parallelism factor P we will always maximize parallelism, SCAN will evaluate the tradeoff between waiting for resources to become free so a new job can run with more cores, compared to starting running sooner with less resources.

The reward scheme just described is an adaptation of our previous work on optimal sizing of biological analysis tasks [**4**]. That work examined how best to execute a variable biological analysis workload when the number and nature of the worker machines is under SCAN's direct control; here we adapt that scheme to an external controller such as CELAR.

Note that whilst this reward scheme requires the administrator to have some expertise in the performance characteristics of the job classes they expect to run using SCAN, the user is only required to provide a size estimate per job, which is much less difficult. For many file formats it suffices to count bytes or records to provide a useful estimate of processing difficulty.

2.4.2 Interaction with the Supervisor

Ideally, SCAN wants its supervisor to provide workers in a quantity appropriate for the current workload, and with "hardware" specifications appropriate for the tasks that SCAN has to schedule. The workers should be sufficiently multicore that the scheduler does not frequently have to run jobs suboptimally because no sufficiently 'large' slot is available, but not so large that there is significant wastage due to partially occupied workers. They should also have the right CPU-cores-to-main-memory ratio that the limits on both are typically encountered simultaneously.

SCAN exposes several statistics to its Supervisor to show what performance bottlenecks are present and direct performance improvement:

- **Queue length**. The number of jobs waiting in the queue for resources to become available, or for the supervisor to supply more workers. High values indicate that adding more workers is desirable.
- **Total reward earned**. The sum of the Reward earned so far from jobs that have completed.
- **Reward lost due to queuing**. The sum of Reward lost so far due to jobs having to spend time in the queue. Gives an idea of the scale of the queuing problem when considered relative to the total reward.
- **Reward lost due to scaling problems**. The sum of Reward lost so far because a job ran with a lower degree of parallelism than is ideal, because no workers had enough cores or only partially-occupied workers were available. If significant compared to total reward, this indicates supplying workers with more cores in the future may be beneficial.

- **Worker CPU core utilization** and **memory utilization**. If both are low, and the queue length is zero, indicates idling workers that may be best released to save money, or if no worker is completely idle, indicates it may be beneficial to hire more workers with less cores, so that fragmentation does not lead to wastage. If one is full but the other is not, indicates the resource balance is suboptimal and we may be paying for one or other resource we cannot use. Note SCAN reports CPU and memory *reservations* (e.g. SCAN has assigned 2/4 cores to running jobs) rather than actual usage, on the assumption that any reservation underutilization is transient and does not represent an opportunity for opportunistic multitasking.

The Supervisor can also advise SCAN that its resource demands are excessive (e.g. we are already spending most or all of our available budget to hire workers). In this situation it is desirable for SCAN to run tasks single-threaded even if this results in lower reward, because single-threaded execution usually has lower overhead, leading to higher long-term throughput and more effective clearing of a long queue.

SCAN maintains a **threshold** expressed in number of concurrently hired cores, above which it will only run jobs single-threaded for this reason. The Supervisor can request that the threshold be lowered by some factor, and SCAN will raise it by a constant increment per job completed. In this way, SCAN will try to gradually relax the single-thread constraint until it is notified again to economise.

2.4.3 Dynamic Profiling

Whilst the SCAN Administrator provides the initial parameters used to calculate the reward to be gained from running jobs, the relationship between users' size estimates, number of cores allocated and completion time is dynamically profiled by the Scheduler and used to improve its model for future tasks.

Once at least 5 jobs of a particular class have completed, linear regression is used to determine the constant time factor involved in running that kind of job and the gradient that depends on the data size estimate provided by users.

Once at least 5 jobs of similar size but different degrees of multithreading have been executed, the Scheduler extrapolates their completion times to find the theoretical completion time of a job with an infinite number of cores assigned. Comparing this to the completion time observed with a single core assigned allows the Scheduler to update its view of the scalability factor which divides the single-threaded runtime into a perfectly-scalable and a non-scalable proportion.

Clearly these linear models could be improved upon; however, we find that they are adequate for modeling the kinds of biological analysis workloads we are interested in.

2.5 Workers

SCAN Workers run jobs assigned to them by the Scheduler. It may assign more than one job to execute concurrently on the worker, and it informs the worker of how many CPU cores and how much memory is assigned to each job as it begins execution.

The worker exposes the number of CPU cores and gigabytes of memory it currently has free to the Supervisor. The Supervisor may use this information to choose, when decommissioning workers to shrink the worker pool, which one is best removed.

2.6 Distributed Filesystem

The SCAN Scheduler and Workers together implement a distributed filesystem for users' Jobs to store their inputs, outputs and temporary data. The workers pool their local storage, which may be adaptively resized by the Supervisor, whilst the Scheduler maintains the namespace, coordinates data movements and arranges transfers between workers when required.

To use the distributed filesystem (SCANFS), users' jobs must specify the paths to their required input files and the outputs they are expected to produce. The Scheduler takes data locality into account, preferring workers that already have, or are already receiving a copy of needed input files. If this is no object, it also tries to achieve balance by assigning jobs to workers with the higher amount of free local space.

In a typical use case, a user with a pipeline of jobs will upload their input data to SCANFS, each pipeline stage will consume either the input or the previous stage's output, ideally running data-local rather than moving data between workers, before finally downloading the final output.

Workers expose an additional metric to the Supervisor, the amount of local free space, so that if possible the Supervisor can attach more storage to that worker, rather than require it to move data to other workers before it can run jobs.

Files in SCANFS are not initially replicated (it is not intended to provide high durability), but replication may arise when data is copied because data-local execution of some job was not possible. If a particular file's replica count rises above an administrator-specified threshold, copies on the workers with the least free space are deleted.

2.7 Client Utilities

SCAN exposes a graphical web interface that provides job submission and access to any viewers the administrator has configured. It also provides command-line utilities for uploading and downloading to/from the SCAN filesystem and for examining the state of the SCAN Service.

In order to support graphs of jobs with dependencies (for example a pipeline or a distributed computation using a fork/join distribution pattern), SCAN also provides a plugin for the GATK Queue job management framework, which provides dependency tracking against a variety of job execution backends. The user can define a Queue script just as they would for conventional cluster execution, adding SCAN-specific attributes such as a size estimate for the job.

Queue then submits each individual task as a SCAN job and polls for job completion to determine when it can submit dependent tasks for execution.

2.8 SCANv1 vs. SCANv2

SCANv1, the version of SCAN described in D8.2 at the end of CELAR project year 2, prototyped many of the features described here but also had significant shortcomings. The principle differences between that prototype and this year's final prototype are:

- SCANv2 adaptively controls the number of cores assigned to each job and can assign jobs to any free worker. SCANv1 required jobs of a particular class to have fixed resource requirements, and would only execute them on workers belonging to the same class.
- SCANv1 workers were homogenous per class, whereas SCANv2 workers are arbitrarily heterogeneous. This makes it much easier for the supervisor to provide an appropriate balance of worker flavors (virtual hardware specifications).
- SCANv1 required the supervisor to judge horizontal scaling decisions based on crude metrics, such as queue length and worker utilization. SCANv2 implements a more sophisticated reward estimation algorithm that assesses the merits of horizontal scaling, making the supervisor's role easier.
- SCANv1 used a central filesystem (specifically a CIFS server) to allow jobs to consume the results of other jobs. SCANv2 implements a distributed filesystem instead, spreading the I/O load and improving overall performance.
- SCANv1 provided a shared database similar in purpose and implementation to its central filesystem. We found that this was not useful for the workloads implemented against SCANv2, so we did not pursue this angle; however we expect the optimal solution for workloads that communicate via a database to have similar characteristics to the SCAN distributed filesystem.

2.9 Summary

We have presented the design of the SCAN system, in which the SCAN Service provides job management and scheduling, automatically optimizing parallelism and storage locality, whilst a monitoring Supervisor such as CELAR observes its performance and adapts the quantity and quality of workers at its disposal. The system maintains a distributed filesystem for user jobs. Users handle scheduling of graphs of dependent jobs themselves, perhaps using the provided GATK Queue plugin.

3 Implementation

In this section we will give the particular detailed implementation decisions taken in realizing the SCANv2 design.

3.1 Scheduler

The scheduler is implemented in around 3,000 lines of Python, using CherryPy to provide HTTP/RPC support and NumPy to provide mathematical support functions needed in its dynamic profiling code. Its performance metrics are exposed as HTTP GET RPCs whilst HTTP POST RPCs implement user commands, such as job submission, and internal commands such as worker registration and deregistration. All RPCs accept URL-formatted arguments and provide JSON-formatted responses in order to provide maximum interoperability with other programming languages.

The same CherryPy server provides the user interface, implemented using simple HTML forms, and viewer modules, which the SCAN Administrator provides as CherryPy dispatch files.

3.2 Workers

Workers only need to provide a GNU/Linux environment with passwordless SSH access from the Scheduler. This is typically achieved using a deployment script that installs a public key corresponding to a private key held by the Scheduler. Any remote actions that the Scheduler makes on a Worker are executed over this SSH channel, whilst Worker actions targeting the Scheduler (such as registration and deregistration) are HTTP RPCs against the Scheduler similar to user commands.

User software that may need to be installed on the workers should be taken care of by the user-submitted scripts. It will be installed as the unprivileged user account used to execute the jobs themselves.

3.3 Shared filesystem

Besides the distributed filesystem that is provided for bulk user data, a shared NFS volume is also hosted by the Scheduler and mounted on each Worker in order to provide simple sharing of low-volume control data. For example, the Queue plugin that is shipped with SCAN for user job dependency tracking requires a temporary directory shared by all workers to store control information, and synchronizing such files across the distributed filesystem is impractical. Because of the low volumes of traffic, we do not expect this central filesystem to become a bottleneck at reasonable workloads.

3.4 Distributed filesystem

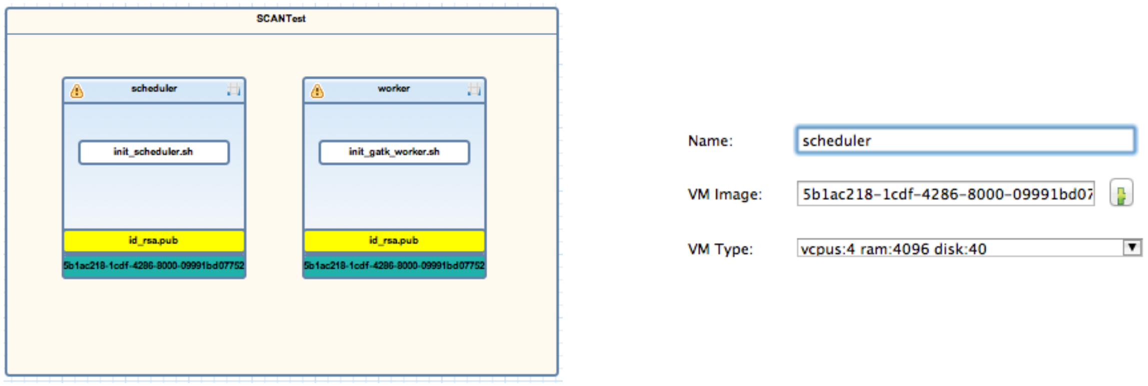
The primary SCAN filesystem is implemented as a simple userspace distributed filesystem with explicit data movement and replication. The Scheduler maintains a map of the namespace that indicates which Workers hold which files, whilst the Workers act as storage nodes.

A Btrfs volume is used on each Worker node to implement local disk spanning, permitting local volumes to be transparently added and removed whilst user jobs are using the filesystem, but the notional shared namespace between nodes is only synchronized when user jobs are explicitly annotated with a need for a particular file. Users specify such files using the symbolic path `/scanfs`, which SCAN rewrites to the local Btrfs volume's real mount point.

For example, a user might implement a two stage pipeline, with a producer that outputs `/scanfs/X` and a consumer that subsequently uses it. They should annotate both the producer and the consumer. The producer annotation updates the central namespace (and triggers a check to make sure the file was really produced); the consumer annotation indicates that the Scheduler should try to find a Worker that already has a copy of the file, or otherwise schedule a data copy to the chosen Worker before the task is permitted to run. The copy takes place between Workers, not via the Scheduler, using `scp`. Keys for copying are distributed by the Scheduler when Workers register themselves.

3.5 SCAN+CELAR Deployment

In order to deploy SCAN with CELAR playing the Supervisor role (a combination I will call SCAN+CELAR for short), we must first describe it using CELAR's Cloud Application Management Framework (CAMF). We create define two "application server" components, one representing the SCAN Scheduler and the other its pool of workers. Figure 3 shows this essential description and the attributes of the scheduler component.

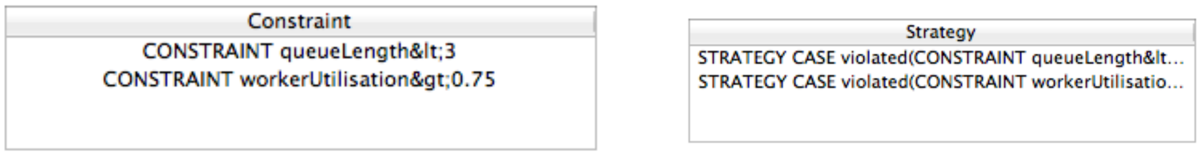


Name:	scheduler
VM Image:	5b1ac218-1cdf-4286-8000-09991bd07
VM Type:	vcpus:4 ram:4096 disk:40

Figure 3: SCAN CAMF description and the Scheduler's key attributes

Next, policy must be declared to guide CELAR in scaling the SCAN worker pool, including picking appropriate virtual machine flavours when commissioning new workers. These are conveyed by specifying constraints (bounds on SCAN's performance metrics) and strategies (procedures to follow if the constraints are broken).

A simple policy set could specify an upper bound on the scheduler's job queue length and a lower bound on overall worker utilization, whilst a more involved set could specify constraints on the reward-loss metrics elaborated in section 2.4. Figure 4 shows the former case, with simple horizontal scaling policy entered into CAMF's GUI.



Constraint
CONSTRAINT queueLength<3
CONSTRAINT workerUtilisation>0.75

Strategy
STRATEGY CASE violated(CONSTRAINT queueLength&lt...
STRATEGY CASE violated(CONSTRAINT workerUtilisatio...

Figure 4: Constraints and strategies for horizontal scaling of SCAN's worker pool

We provided JCatascopia [**5**] probes that convey SCAN's performance metrics to CELAR's decision-making components. These are simple Java programs that request metrics (e.g. queue length, or reward lost due to worker under-provision) from SCAN via HTTP RPCs and forward the results to the JCatascopia Agent.

Finally, we provided shell scripts that initialize the SCAN scheduler and worker (the former at start of day, and the latter whenever a worker is commissioned), and which clean up when a worker is about to be decommissioned or have a storage device added or removed.

Both scheduler and worker initialization scripts assume a basic Ubuntu 14.04 image to start with, and then:

- Install SCAN's dependencies (notably Java, Python and the CherryPy web server)
- Download and build SCAN's JCatascopia probes
- Set up passwordless unprivileged SSH access between scheduler and workers
- On workers only, establish an initial Btrfs volume using a loopback mount of spare space on the VM's main disk, which will host the SCAN distributed FS and may be extended with additional storage later.
- On workers only, download and install software needed to run users' jobs.
- Register workers with the scheduler, marking them ready to enter service.

The script we execute before decommissioning a worker just notifies the scheduler that it will soon leave service (preventing new jobs from being assigned to it) and waits for work already in progress to finish.

The scripts supplied for adding a disk to a worker explore the /dev/disk tree to find the new disk, persistently store the relationship between the infrastructure's device identifier and that seen by the operating system, and add it to the worker's Btrfs volume set. The script for disk removal ensures that all data is moved onto other devices and the device is cleanly removed from the Btrfs volume before the peusdo-physical device can be removed.

This setup, along with an appropriate CELAR orchestrator image for our target infrastructure, permitted us to successfully and fully automatically deploy SCAN on both the Flexiant and Okeanos cloud providers. Figure 5 shows the view of a new deployment against Okeanos as seen in SlipStream, CELAR's low-level commissioning and orchestration system [**6**].

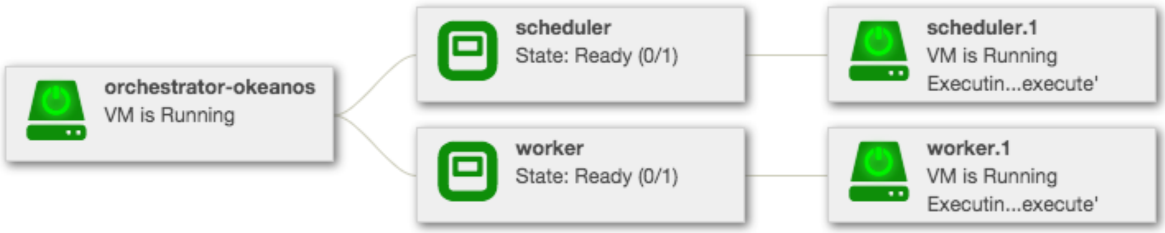


Figure 5: SlipStream view of SCAN newly deployed on Okeanos

4 Example Workload

To evaluate SCANv2, we tested it using three commonly-used biological applications with differing resource needs: the GATK, GROMACS and FIJI.

4.1 GATK

The Broad Institute's Genome Analysis Toolkit [**7**] is a general-purpose suite of tools for manipulating data involved with DNA sequencing. We tested a particular pipeline for calling genetic mutations starting from aligned DNA reads from a sequencing machine. It consists of 7 distinct stages which improve read alignment, adjust the measured read quality scores, call variants and then filter them by statistical confidence.

Each stage of this process exhibits different resource requirements: some are memory-intensive, some are suitable for multithreaded execution whilst others either don't support it at all or don't benefit much, and some have greater storage needs than others. We have previously published a more detailed investigation of the needs of each analysis stage [**3**].

We configured the SCAN service to allocate a class per pipeline stage, so that SCAN would attempt to characterize each stage's behavior individually rather than find a compromise solution for all of them together. We supplied each class with initial parameters (which SCAN uses to predict job execution time based on estimated job size and degree of parallelism) resulting from our previous offline profiling work [**3**]. However since that work used different hardware we expect SCAN's performance to improve once it has observed enough runs to replace those findings with its own dynamic profiling results.

4.2 GROMACS

GROMACS [**8**] is an open source molecular dynamics suite. It is a general-purpose tool for simulating the behavior of complex molecules immersed in a solvent. We particularly use it to determine the equilibrium structures of proteins and thus determine the effects of alterations to their makeup and conditions.

Much like the GATK, it is commonly used to build a pipeline of analysis stages, each of which subjects the molecule being simulated to different conditions and restraints. We assign a SCAN class for each one, and initialize its parameters based on offline testing.

The stages have similar character in terms of their resource demands (requiring considerably more memory than the GATK, but less storage bandwidth) but exhibit differing scalability. This is because whilst the GATK's workloads can often be coarsely divided into well-parallelised sub-problems, GROMACS' problem is inherently monolithic and division into threads is necessarily fine-grained, with a great deal of inter-thread communication. This means that diminishing returns can be experienced at high thread counts, and GROMACS processes can be strong candidates for shrinkage when resources are tight.

4.3 CellProfiler

CellProfiler [**9**] is an open source image analysis toolkit, which we use to automatically locate and classify living cells in digitized images of microscope slides. Unlike the GATK and GROMACS it usually operates single-threaded when run as a batch processing task, and typical task runtimes are much shorter than GATK or GROMACS pipelines. This makes it an ideal "padding" task, able to use spare cores even when fragmented across different machines or only available for a short time.

We configured CellProfiler to convert an input plain image into a coloured diagram indicating where different cells were identified. We vary the analysis difficulty from task to task by submitting different sizes of input image.

5 Evaluation

In this section we present our evaluation of the SCAN prototype, focused on exhibiting its suitability for use with CELAR and identifying directions for improvement.

5.1 Horizontal Scalability

In our first experiment we studied the horizontal scalability of the SCAN resource manager as a whole, running a simple and predictable workload at a variety of worker pool sizes to determine to what degree the centralized aspects of the system limit its ability to scale out.

We ran SCAN under constant load, using a single kind of task (GATK pipeline runs with input size randomly, uniformly distributed between 8 and 12MB). The GATK runs were restricted to run single-threaded at all pipeline stages. The input datasets are far smaller than typical real-world GATK runs, and were picked so that coordination activity at the scheduler was frequent and likely to expose limited scalability in the face of short-running jobs.

All workers in this experiment had 4 vCPUs and 4GB of peusdo-physical memory, hired from the Okeanos cloud service. Neither worker specifications nor task multi-threading were varied in order to avoid extraneous variables perturbing our measurements. With workers of this size, one Queue coordinator task and one GATK analysis can run concurrently on a worker, with memory being the limiting factor. Therefore in order to keep the system busy we ran one client per worker assigned to SCAN, which would repeatedly upload input data, execute the GATK mutation calling pipeline using SCAN and then download the results.

Table 1: Horizontal Scaling Experimental Results

Worker Pool Size	Pipeline Runs Completed	Runtime Mean/SD (minutes)
5	10	7.66 ± 0.48
10	20	8.45 ± 0.59
15	30	10.27 ± 0.41
20	60	10.58 ± 0.81
25	50	12.49 ± 0.48

Table 1 shows the results for between 5 and 25 concurrently-active workers. We observe that scaling is imperfect, with the average pipeline run taking 1.63 times as long when the worker pool and workload are enlarged by a factor of 5. There are many possible factors limiting scalability, such as contention for shared resources at the infrastructure level (for example, worker VMs may share physical hardware, or contend for the same network links), or a reduced chance of achieving data locality in the SCAN filesystem.

Overall however we conclude that this result shows that it is practical to scale SCAN out to moderate worker pool sizes. We plan to extend this experiment to larger worker pools as resource limitations allow.

5.2 Job Duration Prediction

As part of the previous experiment we measured the reliability of SCAN's job duration prediction mechanism (described in section 2.4.3). The scheduler began with a default naïve estimation of each GATK pipeline stage's behavior, but then iteratively improved a linear model relating the job's estimated 'size' (in this case simply measuring the number of records submitted for analysis) to its execution time.

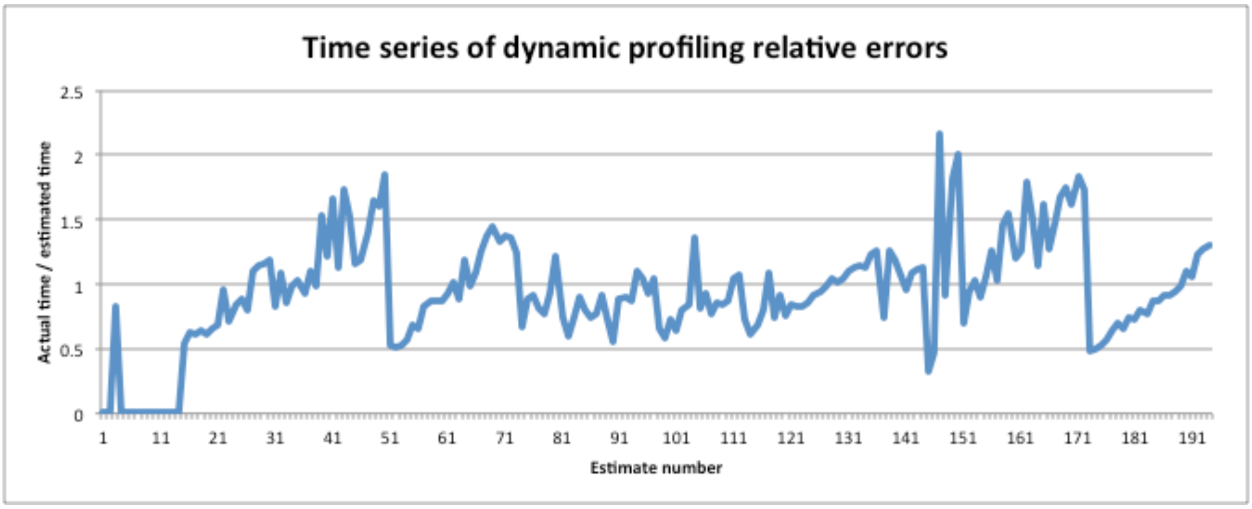


Figure 6: Time series of relative errors in estimating GATK tool runtime

Figure 6 shows the series of estimates made by SCAN's dynamic profiling code in the order they were made, represented as the ratio of actual time to estimated time taken. 1.0 indicates a perfect prediction; 0.5 indicates the actual runtime was half the estimate and 2.0 indicates the opposite. This particular series of estimates was generated in the previous experiment's 25-worker run. The initial series of very bad estimates corresponds to a time when one or more GATK tool types has not been run enough times to train SCAN's model of that kind of task and very pessimistic estimates are produced. The broad trend for actual-to-estimated ratios of around 1.0 is good news, showing that SCAN can generally predict how long a task will run quite well, which is important for its reward-estimation mechanism described previously. The times when a short-term trend in estimate quality, such as the slope from estimate #174 onwards, most likely indicates a period when resource contention is varying, and the model is taking time to adjust to the change in circumstances.

5.3 Job Parallelism

In our next experiment, we subjected the same 25 4-core workers, plus 10 extra 8-core workers, to a sustained load of GROMACS analysis requests from 25 concurrent clients. The workload consumed all available cores, and so SCAN allocated varying numbers of cores to each pipeline stage as it reached the front of the queue in response to presently available worker cores. We measured the relationship between average completion time for each GROMACS pipeline stage and the number of cores it was allocated.

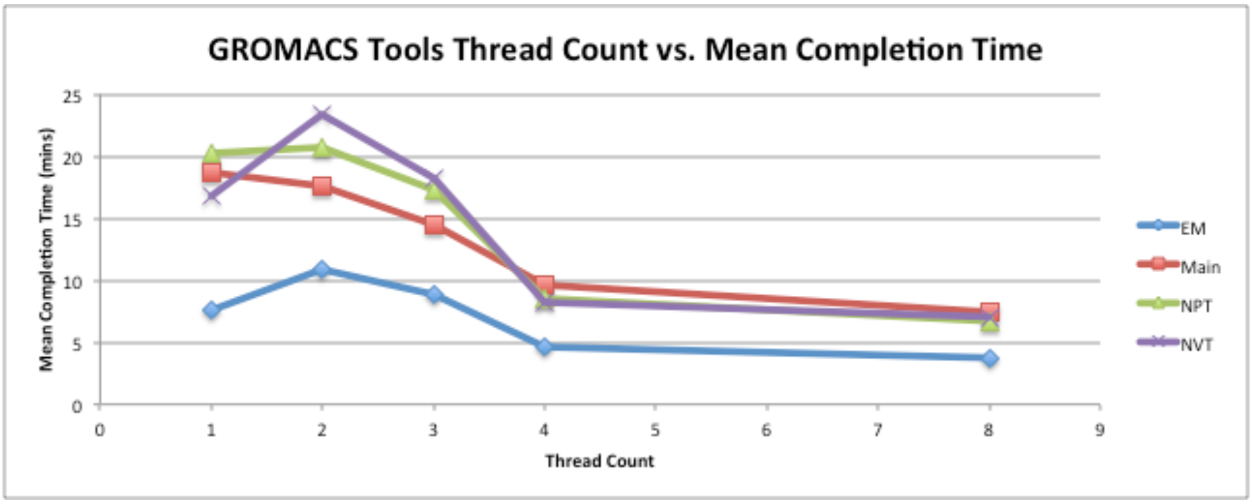


Figure 7: GROMACS performance vs. thread count

Figure 7 shows the relationship observed for each tool (EM, NVT, NPT and Main are stages of a typical GROMACS analysis pipeline, each of which simulates atomic and molecular interactions under differing constraints). As expected the trend is broadly for more threads to improve performance, but we see some interesting effects:

Two cores do not improve performance as much as we would expect, and for two tools actually harms performance. This may indicate that when two GROMACS processes are sharing (virtual) hardware the 3-cores vs. 1-core configuration performs better than the 2-cores vs. 2-cores one, perhaps because the latter involves two processes with conflicting inter-core traffic. However if cache behavior is indeed to blame this clearly depends heavily on the underlying implementation of the virtual machines provided as SCAN workers. This indicates that it may be necessary to improve upon SCAN's current simple model of tasks' degrees of parallelism (described in 2.4.3), as it is not currently able to model this counterintuitive result.

We also observe that moving from 4 to 8 cores has only slight benefit, indicating that if sufficient work is available, better overall throughput would be achieved by running each individual task with less threads. This indicates that there is much to be gained from SCAN's dynamic adjustment of per-job parallelism based on overall workload, as for this particular workload SCAN's users will clearly be happiest with 8 threads assigned per task if SCAN has many idle workers, but with 4 threads per task if there are no spare resources and a queue is building up.

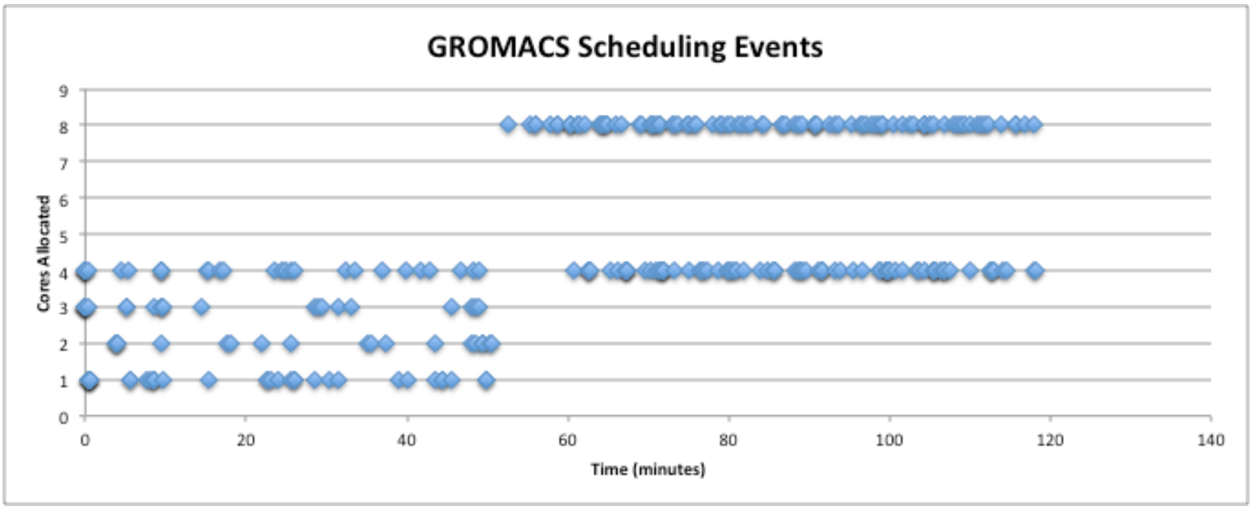


Figure 8: Number of cores allocated to each GROMACS task

We also plot (in Figure 8) the number of cores in SCAN's pool allocated to each GROMACS task as a function of time, showing the effect of adding the 10 8-core workers midway through the experiment. Before they are added tasks are often assigned odd numbers of cores in order to fit them into spare space rather than wait; after they are added resources are more plentiful and all tasks are assigned an entire worker, getting either 4 or 8 cores depending on the best worker available when the task reached the front of the queue.

This illustrates that SCAN's dynamic core allocation mechanism is correctly flexing between compelling tasks to share when resources are scarce towards the beginning of the experiment and permitting them to use all available resources when they are plentiful towards the end.

6 Conclusion and Future Work

We have described the design and implementation of SCAN, a resource manager designed for the elastic cloud, managing a pool of dynamically allocated, heterogeneous workers used to execute scientific analysis tasks. It extends the prior art of resource managers (RMs) by marrying the dynamic job-sizing features seen in modern cloud-oriented RMs such as Apache YARN, with support for arbitrary scientific analysis applications that are typically used in practical biological laboratories. It also implements a novel model of *user happiness* in order to motivate choices regarding the size of the worker pool used by the RM, and the assignment of worker resources to each task under SCAN's control. SCAN's happiness-modelling algorithm [**4**] and preliminary profiling studies [**3**] have been independently published.

Our evaluation of SCAN demonstrates its horizontal scaling capability as well as its dynamic determination of per-job multithreading, trading off job latency (and therefore individual user satisfaction) against overall throughout (and thus global satisfaction).

SCAN has previously been run under CELAR v0.1[1] exhibiting fully automated horizontal scaling. When the CELAR application framework v0.2 release becomes available in the near future we intend to evaluate the complete feedback loop between SCAN's diagnostic metrics, exposed to CELAR to provide advice concerning the workers SCAN needs to run its analysis tasks, the worker VMs that CELAR chooses to commission as a result, and the resultant performance of users' analyses submitted to SCAN for execution. We also intend to investigate applying prior work in modeling application parallelism to improve over SCAN's current simplistic view.

Finally we intend to investigate adapting CELAR from its current heavyweight virtual machine model to work with lightweight isolation systems, such as Linux Containers, or cooperative resource sharing systems such as typical cluster schedulers, in order to combine the convenience and excellent efficiency of these systems with the benefits of dynamic scaling.

[1] Described in deliverable D8.2 [**2**]; a video of this deployment is also available at http://cognito.cslab.ece.ntua.gr/final/scanFinal2.mp4

7 Bibliography

[1] Dimitrios Tsoumakos, Stalo Sofokleous, Ioannis Liabotis, Vangelis Floros, Christos Loverdos Wei Xing, "Translational Cancer Detection Pipeline Design," CRUK Manchester Institute, 2014.

[2] Wei Xing, Demetris Antoniades, Andoena Balla, Vangelis Floros, Ioannis Liabotis, Giannis Giannakopoulos, Georgiana Copil Christopher Smowton, "Translational Cancer Detection Pipeline over CELAR Application V1 and Updated Design," CRUK Manchester Institute, 2014,.

[3] Christopher Smowton et al., "Analysing cancer genomics in the elastic cloud," in *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2015.

[4] Christopher Smowton, Georgiana Copil, Hong-Linh Truong, Crispin Miller, and Wei Xing, "Genome Analysis in a Dynamically Scaled Hybrid Cloud," in *IEEE 11th International Conference on eScience*, 2015.

[5] Athanasios Foudoulis et al., "Cloud Information System," University of Cyprus, 2015.

[6] Konstantin Skaburskas, "CELAR System Prototype," SixSQ, 2015.

[7] Aaron McKenna et al., "The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data," *Genome Research*, vol. 20, no. 9, pp. 1297-1303, 2010.

[8] Herman JC Berendsen, David van der Spoel, and Rudi van Drunen, "GROMACS: A message-passing parallel molecular dynamics implementation," *Computer Physics Communications*, vol. 91, no. 1, pp. 43-56, 1995.

[9] Anne E Carpenter et al., "CellProfiler: image analysis software for identifying and quantifying cell phenotypes," *Genome Biology*, vol. 7, no. 10, p. R100, 2006.