Black Hole Effect: Detection and Mitigation of Application Failures due to Incompatible Execution Environment in Computational Grids

Aditya Nishandar, David Levine, Sankalp Jain
Department of Computer Science and Engineering
The University of Texas at Arlington
Arlington, TX 76010, USA
{ nishanda, levine, sjain }@cse.uta.edu

Gabriele Garzoglio, Igor Terekhov
Computing Division
Fermi National Accelerator Laboratory
Batavia, IL 60510, USA
{ garzoglio, terekhov }@fnal.gov

Abstract

Scientific and Engineering domain Applications like high energy physics applications have huge computational and storage needs. Such applications rely mainly on geographically dispersed productions farms for their computing needs. Production farms typically consist of commercial off-the-shelf (COTS) clusters, Network of Workstations (NOW) and intranets. In a grid environment, these clusters are interfaced to the grid middleware via cluster management software like a batch system. In this paper we focus on a detrimental effect that one or more faulty nodes in a cluster have on the efficiency and throughput of the cluster and the application in general. This effect is what we call as the 'Black hole effect'. A 'Black Hole' is a faulty node in the cluster that causes the application to fail on it due to incompatibilities between application requirements and the execution environment.

1. Introduction

A computational grid has been defined as a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high end computational capabilities [7]. Large scale distributed systems like grids have great potential for incurring faults; this has been one of the main hindering factors in the deployment of applications on computational grids [12]. The issues of fault tolerance and reliability in distributed systems have been a significant area of research for many years, because of the high failure rates intrinsic to such systems and a variety of solutions have been proposed to detect and respond to faults and failures in a distributed system in general. The grid fabric consists of a multitude of hardware and software components that are prone to failures. The Grid Fabric layer provides the resources to which shared access is mediated by Grid protocols: for example, computational resources, storage systems, catalogs, network resources, and sensors [8]. Clusters are part of the grid fabric for grids hosting high energy physics applications. Compute nodes of the clusters as well as the interconnecting network are prone to failures. Such component failures can lead to unanticipated, potentially disruptive failure behavior and to service unavailability. Monitoring systems like Supermon [11] and GEMS [12] assist the system administrators in determining failures within a cluster, but the existing grid job management systems cannot recover or mask such failures. Application level failures are presently detected at the best by manual intervention or by the grid job management layer by checking the exit status of the execution returned by the batch system or by modifying the application to deal with failures, which can be a non-trivial task for legacy applications like high energy physics.

Our focus in this paper is determining and reacting to failure caused by the mismatch between the application requirements and the execution host. In particular we deal with failures that occur when the execution host fails to provide a required service or a set of services to the application, resulting in the application crashing. Solutions such as checkpointing, replication and message logging are complimentary to the solution proposed in this paper.

Four broad types of failure classes are defined by [3]. Omission failures occur when a process does not respond to an event. Timing failures occur when a process does not respond before a timeout has expired, or responds prematurely. Crash failures are unrecoverable states when a process completely stops executing. Byzantine [6], or arbitrary failures, is the class of failures that are unable to be accounted for in the design of a system. Fault detection is a prerequisite for recovering from faults. Applications executing in a distributed environment rely on various tools and services provided by the execution environment. Resource allocation and management systems like Condor [10], manage resource allocation by

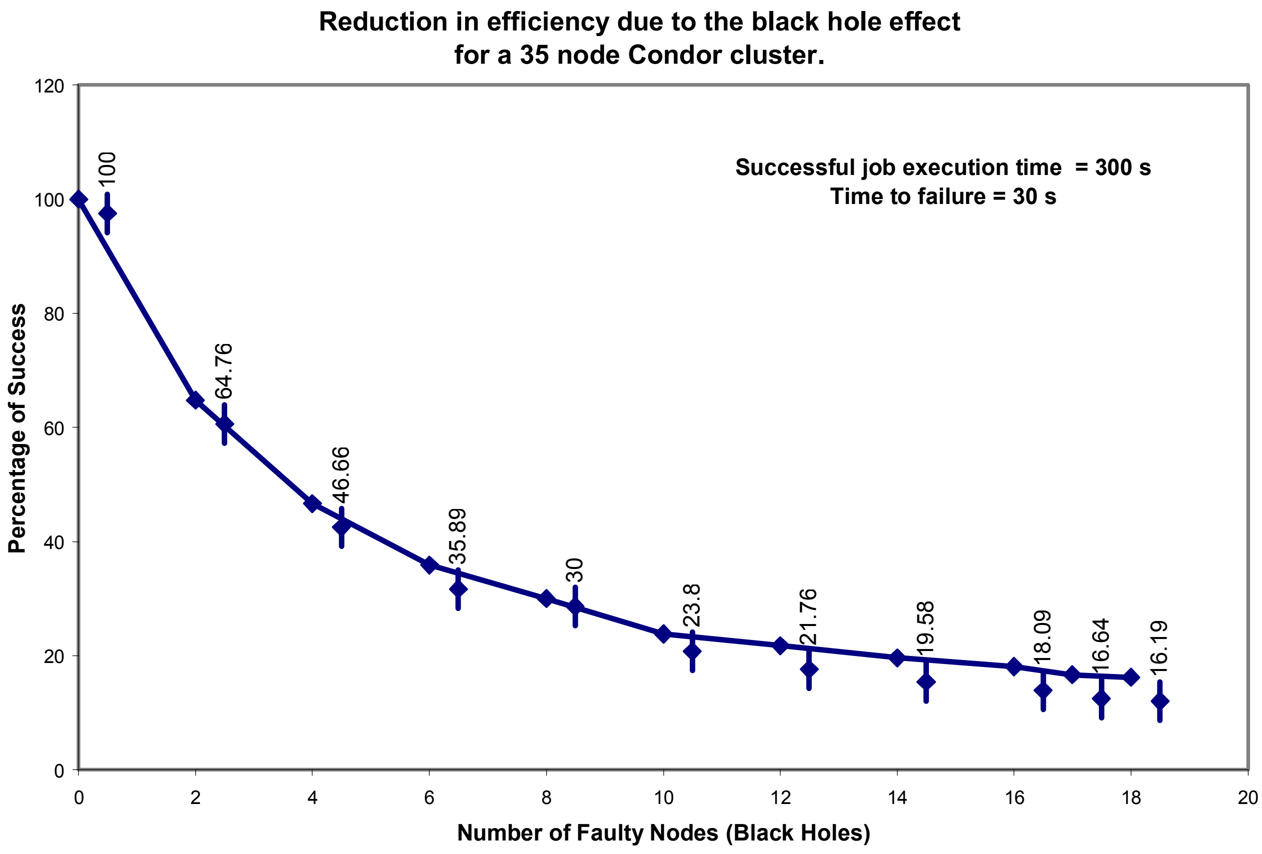

Figure 1. Reduction in grid job efficiency due to faulty nodes

specialized description languages that perform the dual task of describing the available resources at the provider sites and the resources requested by the application. Applications typically rely on a set of basic services offered by the execution environment like compression utilities and system libraries, and usually do not explicitly specify these in the job description to the resource management system. Compute nodes in Beowulf clusters or COTS clusters are heterogeneous in terms of the underlying architecture or the software environment and utilities. Nodes in a cluster typically do not have the exact same set of libraries or utilities. Version conflicts may result in the application failing on a particular compute node, which can potentially have a cascading effect. The entire batch jobs of the same application queued in gets scheduled to the compute node failing the application, resulting in low throughput and efficiency. This phenomenon is what we describe in this paper is called the 'Black Hole Effect'. The term application is used to denote a collection of batch jobs that are queued in the batch system. All the queued jobs are similar in terms of their computational complexities, running time and the execution code. Computational grids provide access to high end computational resources like Beowulf clusters and High Performance Computing Clusters (HPCC), typically to a geographically distributed, diverse user base. HPCC provide a single system image. The application can be executed on any of the servers within the cluster. HPCC typically do not exhibit the black hole effect because of the single system image, they have all or none semantics, i.e. the application can execute with success on all the nodes (servers) or it cannot execute at all. Beowulf clusters are scalable performance clusters based on commodity hardware, on a private system network, with open source software infrastructure. Cluster architectures in which the compute elements are heterogeneous in terms of hardware and software are prone to the black hole effect. In a grid environment, the computational and storage resources are typically in different administrative domains and geographically dispersed. An application wanting to make use of the grid resources has to yield to the local policies of the resource provider and the heterogeneity of resources. The grid software must shield the application from the detrimental effects of the underlying fabric. In the rest of the paper we describe the problem of detecting *Black holes* in more detail and propose three different classes of approaches of mitigating its effect on the efficiency of the applications.

We frequently cite the problem and its resolution in the context of the SAM-Grid [1] [9] infrastructure, a distributed system for job, data and information handling for high energy physics applications. Section 2 elaborates on the concept of *Black holes* and their how the efficiency of an application exacerbates in a grid environment by taking Monte Carlo simulations on the grid for high energy physics (HEP) as an example application. Section 3 describes observed scenarios of how black holes come into existence and their detection. Section 4 outlines different approaches to deal with the effect. We conclude with section 5 and provide future direction for research in section 6.

2. What are black holes?

Monte Carlo methods are widely used in scientific computing and simulations. Applications using Monte Carlo methods, like High Energy Physics (HEP) applications account for a major percentage of the applications running on widely deployed grid infrastructures like SAM-Grid. Monte Carlo applications are usually compute intensive operating on data sets of varying magnitude in terms of size. Parallelism is a way to accelerate the convergence of a Monte Carlo computation. If N processors execute N independent copies of a Monte Carlo computation, the accumulated result will have variance N times smaller than a single copy. Grid jobs in production grids like SAM-Grid manifest themselves into multiple local jobs at an execution site. A grid job is a description of the user requirements for the application (Monte Carlo simulation). For example for a typical DØ [2] Monte Carlo simulation grid job, the user specifies the number of physics events to be simulated and the control parameters for the simulation. A single grid job is mapped to multiple local jobs after the resource allocation is done by the job management system. Local jobs operate on a smaller chunk of the grid job. When the grid job is instantiated at the execution site it usually outnumbers the compute elements (nodes), resulting in local jobs getting queued in the batch system. Most of the execution sites have a mix of dedicated and non-dedicated computational resources, managed by a network batch queuing system like Condor and PBS. The batch system schedules these local jobs on compute resources as they become available. The application relies on some basic services provided by the compute nodes, if any of these required services malfunction; the application fails and exists the compute node with an error status. The batch system is unaware of the cause of the failure and schedules another job of the same application to the same compute node which is bound to fail. The batch system continues to schedule jobs on the same compute node causing spurious high turnaround, and eventually yielding very low throughput for the grid job. This effect is called the *Black Hole effect*, and the compute nodes which malfunction with respect to the application are called *Black holes.*

3. Black hole detection.

Figure 1 represents the decrease in efficiency of a grid job. The tests were carried on a cluster having 35 compute nodes, running Red Hat Linux 7.2 and managed by Condor batch system. A total of 210 jobs were submitted for each configuration of black holes. The batch system had no other jobs except the jobs submitted for the test. In the rest of this section we provide a more insight into the detection of *Black holes.*

When jobs exhibit high turnaround on a particular node, how should the grid software determine if the jobs are really failing on that node, or the node is just a combination of powerful processing element, large memory?

Black hole detection is aided by the application specifying a minimum duration for the job to execute (d_{min}). From empirical data d_{min} can be estimated within an error margin. d_{min} is expressed in terms of wall clock time for a given architecture and a given operating system. For example to simulate 250 physics events of a DØ Monte Carlo simulation job, it takes at a minimum 8 hours (d_{min}) between the time it is scheduled by the batch system and the time that the error and output logs are returned. The reference architecture is an Intel x 86 with 512 mega bytes of physical memory, 1024 mega bytes of virtual memory, 1GHz of clock speed, running a Linux 2.4.x kernel.

In determining if a node is really a black hole, we normalize the values of the compute nodes processing power (processor) and its memory with respect to the reference system, thus arriving at a normalized value of $d_{min.}$ Empirical evidence suggests that if the amount of time taken by an execute node to complete a job is less than 60% to 80% of the d_{min} value, (leeway of 20% to compensate for mathematical approximations), then we have a potential black hole.

How does the grid software react, if we have mixture of local jobs having a relatively high variation in their execution time?

The Black Hole effect is an application specific effect. A node which is designated as a black hole for one application might be a perfectly fine node for some other application if that application does not rely on the same services for its execution. For example if a Bio informatics application does not use a particular system library or is able to handle inconsistencies in the version of a system library, then that application can execute fine on a black hole for a high energy physics application.

Which layer of software detects the presence of Black holes?

The job management software on the execution site like the Globus jobmanager can detect these faulty nodes, since a jobmanager is started for each grid submission. We propose the detection of *Black holes* as a part of the grid-fabric interface [4]. In SAM-Grid, the grid-fabric interface is a collection of services on the execution site gateway. The gateway interfaces the grid middleware to the batch system, by dynamically instantiating the job management service. The job management at the execution site is assisted by these services. These services try to mask the failures and the shortcomings of the fabric.

What fabric services are necessary in order to detect black hole?

Some basic services are expected of the fabric to detect the existence of a black hole. These services are incorporated in many popular batch queuing systems. Service like retrieval of the standard output and standard error streams of the finished jobs. API's for explicitly specifying the compute node on which a job should execute.

We now provide with some observed scenarios. These scenarios are based on practical experiences and by no means are complete or exhaustive, but stem from our experience of executing physics applications on production grids

- Grid software which depends on authentication mechanisms via X509 certificates exhibit failures due to clock skews, especially if the clock is slow. The X509 proxy credentials will not be valid if the system clock is slow, thus resulting is failure of accessing grid services relying on X509 authentication mechanism from the compute node in the cluster. If the application scheduled on a compute node with a slow clock utilizes the X509 proxies for accessing grid services, then the application will fail on that particular compute node.
- Inappropriate software. If a job expects some basic tools, usually provided by the operating system such as compression utilities et cetera, and these basic services are not present or behave inconsistently, it results in the application failing. For example applications which depend on GNU-zip are susceptible to the black hole effect if the compression utility malfunctions or has the incorrect version.
- *hostname* utility, or in general gethostbyname() does not return the fully qualified domain name, this might result in the failure of network services which might work locally, but fail in a grid environment.
- Conflict with system libraries. Applications relying on system libraries for their execution could fail if these libraries are not compatible with the application. This is especially observed with the glibc library in flavors of Linux for dynamically linked code.

4. Reacting to the black hole effect

We have described some potential scenarios of how black holes might come into existence. We now present a couple of approaches to recover and whenever possible avoid the black hole effect.

4.1. Proactive probing for black holes

The proactive approach of determining whether a compute node can be a potential black hole relies on the application describing in a formal way as to what services it expects from a compute node in the batch system. This approach is a pessimistic approach in the sense that it assumes that there will be potential black holes in a cluster and tries to avoid them by isolating these nodes for the application. These services can be specified via a service description language. The grid software then has the responsibility of sending small probes (test jobs) to all the nodes in the batch system and analyzes the results of these probes. Persistent information regarding the total number of reachable nodes, nodes suitable for executing the application based on the criteria (good nodes) and nodes unsuitable for executing the application (potential black holes) is maintained.

Pros:

- A correct formal description of services required from the compute node is the only requirement of this approach.
- Can easily be accommodated into an existing Grid infrastructure, with out changing a significant amount of application code.

Cons:

- Is highly dependent on the correct formal specification of the services required from the compute nodes.
- The persistent information regarding the state of the cluster is not very dynamic and might result in wasted resources.
- Relies on the fact that there is some non trivial way to get a list of all available compute nodes.
- Populating the persistent storage might be a very time consuming task and the information collected might be outdated and resource wastage might occur. For example in a Condor pool of computing resources, the state of the Condor pool is constantly changing. New nodes become available and existing jobs might be checkpointed and/or preempted during their run, thus making the whole environment highly dynamic.

4.2. Reactive approach - Mining the logs

The reactive approach for dealing with the black hole effect is an optimistic approach. We start by assuming that all the compute nodes in the cluster comply with the applications service requirements. This approach is at the best probabilistic and depends greatly on the availability of statistical data regarding the behavior of the application. For example the grid software needs to know what is an approximate time that the application takes for an Intel x 86 machines with 512 mega bytes of physical memory, 1024 mega bytes of virtual memory and the processor clock speed of 1 GHz. The values can then be normalized depending on the similar parameters of a compute node. When a grid job is instantiated (manifests itself into multiple local batch jobs), we let the grid software submit multiple jobs to compute nodes in the batch system from a list of available nodes. Each node in the list is initially assigned a weight of 0 units. A sorted list of nodes is maintained with the nodes having higher weights at the head of the list. If a job successfully executes on a node its weight increases exponentially. If a job fails to execute on the node, then its weight reduces exponentially. This ensures that the most of the jobs are scheduled on good nodes at the same time avoiding the starvation of nodes. The jobs are scheduled by the batch system in from the head of the list, thus increasing its chances of success.

Pros:

- Very dynamic in nature. Gives high priority to the latest results.
- Needs no formal specification of what services the compute node should provide.

Cons:

- Determining the cause of is difficult, and mining of log files is necessary.
- Depends on the fact that batch system provides a service of submitting jobs to selective nodes. Also if a list of nodes is given, the scheduling of jobs on these nodes might be biased.
- Need complex statuses or messages to determine if the failure was due to the application failing (property of the application) or one of the services required by the application failing (property of the node).
- Complex data structure required to maintain information, overhead might not be trivial.
- Collection of statistical information might be an overhead.
- Normalization needs to be done depending on the characteristics of the compute node and the characteristics specified by the application, which might not be trivial.
- Since this approach allows failures and learns from them, the efficiency of the grid infrastructure is hampered to a non negligible extent.

If a node executes a job successfully, then its weight increases rapidly during the initial few successes and comparatively slowly for the later successes, thus following a logarithmic trend.

If a node having positive weight fails to execute a job, we reduce its weight linearly until it reaches the initial weight of 0 units. After this point the weight reduced exponential. Thus if a node is a potential black hole, the first job scheduled to it will fail and it will immediately receive negative weight. On the other hand if a node with positive weight fails a job, we classify this kind failure as Byzantine failure [3] . The higher the weight associated with a node, the higher is the probability that jobs will execute successfully on that node if scheduled at some later point in time.

4.3. Hybrid approach - Probing with Mining

The Hybrid approach tries to incorporate the advantages of both the previous approaches. A periodic probing of the nodes combined as in the proactive approach combined with the dynamicity of the reactive approach would be ideal. A list of good nodes can be built by probing and this information is periodically refreshed depending on the outcome of the local jobs as in the reactive approach. The hybrid approach offers the best of both worlds, but the complexity of implementation might be the only hindering factor in its deployment.

4.4. Black holes and system throughput

Throughput is output relative to input; the amount passing through a system from input to output.

T_{fail} = Time taken by an application to fail on a compute node.

$T_{success}$ = Time taken by an application to complete successfully.

Failure rate = Number of failed jobs per unit time.

Throughput = (Successful jobs / Total number of jobs).

A batch system with 35 execute node is submitted 210 jobs. $T_{success}$ is 100s and T_{fail} is 2s. If we have 2 black holes, 1 job fails per second (Failure rate = 1 jobs per second). Success rate is 33 jobs per 100s, i.e. 0.33 jobs per second. At the end of 100s the throughput of the system is 24.81. The existence of black holes increases the input to the system with out producing any useful output, thus adversely affecting the systems throughput. The greater the time to fail for a job on a node, the greater is the throughput of the system.

Throughput is inversely proportional to the number of black holes and T_{fail}.

5. Related work

Fault detection and response has been a highly researched topic for the past decade or so. [12] Propose a fault detection service for wide area distributed computing environments, like computational grids. This service uses well-known techniques based on unreliable fault detectors to detect and report component failure, also known as fail-stop failures, while allowing the user to trade of timeliness of reporting against false positive rates. Many batch queuing systems like Condor support checkpointing and migration, message logging to deal with application failures. These mechanisms would improve the efficiency when recovering from failures due to *Black Holes*.

6. Conclusions and Future Work

We have described the problem of applications failures due to incompatible execution environments. We also demonstrate how these failures exacerbate in computational grids as compared to isolated clusters. Different approaches have been proposed to detect and mitigate such failures. In the SAM-Grid system we have been successful in dealing with the *Black hole effect* by providing fault tolerance in the grid-fabric interface [5]. Majority of the reasons why applications fail on the worker nodes is because of the heterogeneity of the cluster and lack of formal specification as to what services the application expects from compute resources. Such formal specification is not always possible due to the fabric limitations and hence the need for shielding grid applications from faults.

7. Acknowledgements

This work was partially supported by the Department of Computer Science and Engineering, Department of Physics at The University of Texas Arlington, TX, USA; and Fermi National Accelerator Laboratory, Batavia, IL, USA, FNAL-PO-546763 and conducted with the United States Department of Energy SciDAC program, the Particle Physics Data Grid (PPDG), and the Grid for UK Particle Physics (GridPP).

8. References

[1] A. Baranovski, G. Garzoglio, K. Koutaniemi, L. Lueking, S. Patil, R. Pordes, A. Rana, I. Terekhov, S. Veseli, J. Yu, R. Walker, V. White, "The SAM-GRID project: architecture and plan", Nuclear Instruments and Methods in Physics Research, Section A (Elsevier Science), Proceedings of ACAT'2002.

[2] DØ experiment: http://www-d0.fnal.gov

[3] F. Christian, "Understanding Fault-Tolerant Distributed Systems," in Communications of the ACM, 34(2):56-78, 1991.

[4] G. Garzoglio, I. Terekhov, A. Baranovski, S. Veseli, L. Lueking, P. Mhashilkar, V. Murthi, "The SAM-Grid Fabric services", talk at the IX International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT-03), Tsukuba, Japan, Dec 2003.

[5] G. Garzoglio, I. Terekhov, J. Snow, A. Nishandar, S. Jain, "Experience producing simulated events for the DZero experiment on the SAM-Grid", presented at Computing in High Energy and Nuclear Physics (CHEP 2004), Interlaken, Switzerland, Sep 2004.

[6] H. Attiya and J. Welch, "Distributed Computing Fundamentals, Simulations, and Advanced Topics", John Wiley and Sons Inc, ISBN 0-471-45324-2.

[7] I. Foster and C. Kesselman (eds.), "The Grid: Blueprint for a new Computing Infrastructure", (Morgan Kaufmann Publishers, 1998) ISBN 1-55860-475-8.

[8] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the grid. Enabling scalable virtual organizations", International Journal of Supercomputer Applications, 2001.

[9] I. Terekhov, A. Baranovski, G. Garzoglio, A. Kreymer, L. Lueking, S. Stonjek, F. Wuerthwein, A. Roy, T. Tannenbaum, P. Mhashilkar, V. Murthi, R. Walker, F. Ratnikov, T. Rockwell, "Grid Job and Information Management for the FNAL Run II Experiments", in Proceedings of Computing in High Energy and Nuclear Physics (CHEP03), La Jolla, Ca, USA, March 2003, La Jolla, California, March 2003.

[10] M. Litzkow, M. Livny, and M. Mutka, "Condor - A Hunter of Idle Workstations", Proceedings of the 8th International Conference of Distributed Computing Systems, pages 104-111, June, 1988.

[11] M. Sottile and R. Minnich, "Supermon: A High-Speed Cluster Monitoring System", in Proceedings of IEEE International Conference on Cluster Computing, 2002.

[12] P. Stelling, et.al., "A Fault Detection Service for Wide Area Distributed Computations", Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, 1998.

[13] R. Subramaniyan, P. Raman, A. D. George, and Matthew Radlinski, "GEMS: Gossip-Enabled Monitoring Service for Scalable Heterogeneous Distributed Systems", submitted to the journal of Network and Systems management.