Applying Internet Random Early Detection Strategies to Scheduling in Grid Environments

Steven Willmott¹, Manuel Brugnoli², Elisa Heymann², Paul Hurley³ and Miquel A. Senar²

¹ Universitat Politècnica de Catalunya, Spain.

steve@lsi.upc.edu

² Universitat Autònoma de Barcelona, Spain.

manuel.brugnoli@aomail.uab.es,

{Elisa.Heymann, [MiquelAngel.Senar](mailto:MiquelAngel.Senar@uab.es)}@uab.es

³IBM Research, Zurich

pah@zurich.ibm.com.

Abstract. Resource Allocation in Grid environments to date is generally carried out under the assumption that there is one primary scheduling system scheduling jobs. However, as environments tend towards larger open “utility” Grids it becomes increasingly likely that deployments will involve multiple independent schedulers allocating jobs over the same resources.

In this paper we show that, if using current standard scheduling approaches, such multi-scheduler environments may well be prone to serious oscillation problems in resource allocation similar to those commonly found in IP network traffic. Further we demonstrate how common techniques from IP networks – in particular approaches based on Random Early Detection (RED) buffer management and its subsequent extensions / variations – may provide an effective way to damp or eliminate such oscillations. The paper describes the analogy between multi-scheduler Grid resource allocation and IP network routing and explores the impact of oscillation and RED methods by simulation.

1 Introduction

Grid computing environments are a powerful model for virtualized resource sharing in wide area network environments - making it possible for many users from different organisations to transparently access computing resources. However, the resulting resource allocation problem is highly challenging and becomes particularly so when:

- Processors are widely distributed – leading to lengthy communication delays between sites and resource controllers having to work with outdated load information.
- Users are associated with multiple centres of authority and are also distributed – making it impractical to centralise resource scheduling for scalability, performance and administrative reasons.

In such environments, resource scheduling involves multiple independent schedulers, each serving a subset of users and able to place jobs on resources without direct communication with other schedulers. Essentially the schedulers are multiple actors working over the same resources in parallel with no central oversight. Furthermore, the information the schedulers have to work with is time delayed and potentially inaccurate.

While this situation is not commonly found in Grids today, it represents a scenario which is increasingly likely for large-scale open “utility” Grid type environments in which users may be accessing Grid resources through multiple, administratively distinct access points (much as they access Internet communication capacity today). Furthermore, even in small Grid deployments (between different departments at a single university for example), having to deploy one centrally administered scheduling system is often an undesirable limitation.

For this reason, multi-scheduler environments in which each scheduler acts independently arguably represent an important class of problems to be addressed for future Grid environments. However, such problems are arguably structurally very different to those in which a single actor (a single scheduler or a hierarchically organised cluster of schedulers) can be assumed. In particular, in this paper we argue that environments with multiple independent scheduling processes working over the same set of resources can potentially exhibit some of the same types control problems commonly found in today's packet switched IP (Internet Protocol) networks – namely pathological oscillations in resource allocation caused by unintended synchronisation of job allocation by independent actors - leading to significant inefficiencies in performance.

While great advances have been made in resource scheduling in recent years (from resource controllers [1, 2, 3, 4, 5, 6] which now work very effectively in environments where the scheduler is the primary actor to approaches such as application based scheduling [7]) it may be non-trivial to extend these techniques to larger, multi-

scheduler environments. In particular, while there has been work on multi-scheduler environments (for example on coordination via hierarchies in [8], on agent based approaches [9] and to an extent on market based approaches [10]), there has been little detailed analysis of the dynamics of such environments.

In this paper we suggest that multiple scheduler Grid environments do indeed show potentially complex and pathological dynamics and further that one way to tackle such problems may be to borrow techniques from IP networks themselves. In particular intelligent buffer management techniques such as Random Early Detection (RED) [11, 12] and (later) [13, 14] may provide powerful models for scheduling in future Grid Environments. In this paper we:

- Show how Grid resource management involving multiple schedulers could be beneficially modelled as IP network buffer management.
- Demonstrate that at least under naïve implementations of common scheduling techniques, pathologies such as oscillations may well arise.
- Present two RED based strategies (Static RED and dynamic RED) for resource scheduling which could be used to mitigate such behaviour.

Results show that under certain conditions with multiple schedulers, oscillations clearly exist and heavily impact resource usage and that RED, in particular when augmented to function with dynamic thresholds can very significantly impact in terms of performance improvement when applied.

The paper is organised as follows: Section 2 provides an abstract definition of the Grid scheduling problem addressed, Section 3 describes the potential dynamics of such systems, Sections 4 and 5 provides experimental results and Section 6 discusses the potential implications of the work. Section 7 concludes the paper.

2 Problem Definition

In order to draw on techniques from network buffer management we define a generic multiple scheduler Grid resource allocation problem as follows. The world consists of:

- A set of n_s schedulers s_i *in* S which do not communicate with each other and each of which emits a series of jobs $(j_{i1} .. j_{in})$ *in* J over time and aims to assign them to processors.

- A set of n_r processors p_i in P , each of which has a processing queue with limited capacity $pcap$ and is able to (on request) send out information on the status of its current processing queue (length in items or time) $pinf$.
- Processors handle jobs in a FIFO (First In First Out) manner. Once they have accepted a job it is always eventually processed (processors are assumed not to fail).
- Each scheduler regularly polls all processors to get the latest $pinf$ information and caches this between updates.
- For simplicity, the time delay t for a message to travel from any s_i to any r_j and back again is the same and is non-zero. (Hence, if an s_i sends a request to an r_j at time T_i , the r_j will receive the message at time T_i+t and s_i the response at T_i+2t .)¹
- For a scheduler s_i to assign a job to a processor p_j it sends a message to the processor placing the job. The resource then responds either with an *acceptance* of the job or a *rejection*.
- Once a job has been assigned to and accepted by a processor it cannot be re-assigned. If the job is rejected by the processor the scheduler may try to assign it elsewhere.

In this environment, it is therefore assumed that each scheduler uses the polling mechanism to produce a regularly updated snapshot of the current resource state of the network. The scheduler subsequently makes individual decisions on which processor to place its own incoming jobs. No direct communication is assumed between schedulers since such communication would take at least as long as interactions with processors and generate a strong synchronisation dependency on schedulers.

3 Oscillations and RED in IP Networks

Communication networks based on the Internet Protocol (IP) and packet-switched networks more generally function based on a network of nodes (routers, switches) which each have a limited buffer capacity to hold data packets (chunks of data) as they are routed onwards to the next node in the network. Data transfer is carried out between a source and an endpoint (in multicast cases, multiple endpoints) as a stream of independent packets and files are reconstructed from their constituent packets at their destinations. Flow control mechanisms such the TCP/IP protocol [15] provide

¹ More realistically the delays between each resource and scheduler would be determined by the network topology and would not always be the same.

signalling to communicate packet arrivals at the receiver to the sender. In the case of congestion (overloaded nodes at any point on the path between the sender and the receiver) packet losses occur, which induce a “backing-off” (reduction) of the sending rate at the sender. The sender then slowly increases its sending rate once more.

As a further layer on top of basic traffic send-rate mechanisms, routing protocols (such as RIP based routing [16]) operate in the background to set the shortest routes between endpoints. Dynamic routing mechanisms are also used to potentially switch routes to avoid congested areas of the network (such as OSPF [17]) – thereby dynamically re-allocated traffic flows across other resources in the case of congestion.

These basic mechanisms make it possible for all possible endpoints in the network to open and close TCP/IP packet flows to other endpoints without direct communication with each other. In other words, there is no central resource control. Furthermore, congestion is managed using feedback signals generated by the TCP/IP protocol (or similar mechanisms) which indicate to senders concerned when congestion occurs. These mechanisms provide for a baseline in congestion management. However, unfortunately they also exhibit pathological properties under certain (common) network conditions. In particular, the two most serious issues are:

- **TCP/IP Backoff synchronisation [11]:** this phenomenon occurs when one or more nodes in a network become congested and the TCP protocol signals all senders with streams passing through the congested area to “slow down” sending within a short space of time. The result is a quick drop in all traffic through the node (creating a hole in resource use) followed by a gradual speed-up of sending by all sources (potentially creating a further congestion peak, another hole and so on).
- **Route Oscillation:** in networks which use dynamic route allocation and adjust paths through the network based on round-trip times, congestion may cause routes to change to avoid a congested area. However, in general this means changing all traffic between two endpoints – hence, such route shifts may therefore create congestion elsewhere, a consequent shift back to the original route and so on.

These phenomena are well documented in the communication networks literature (see [18] for example) and have been identified in many complex variations.

While future Grid networks are unlikely to have as many independent schedulers as communication networks have endpoints, as the number of independent schedulers

grows, they may well exhibit similar potential oscillation problems since individual schedulers may naturally be programmed to “back-off” usage of congested processors and/or switch jobs to less loaded machines.

3.1 RED for Oscillation Damping

In the face of these oscillation problems, one of the main solutions developed in communications networks to improve performance is a family of mechanisms based on the idea of Random Early Detection (RED) policies. The underlying mechanism for RED is based on the observation that oscillations are caused by many senders in the network receiving congestion signals at the *same point in time* (or within a short space of time) – in general when the maximum buffer capacities at one or more nodes in the network are reached and all streams receive rejections. RED policies therefore adopt the following underlying principle – *begin sending congestion messages to some endpoints before maximum capacity is reached*. In its basic form, RED applies the following policy to buffers at all nodes in the network [12]:

Two thresholds are set – MinQ and MaxQ as a %age of the maximum physical buffer. For each incoming packet, if the buffer has less than MinQ packets in it, the packet is always accepted, if more than MaxQ it is rejected. For any value between MinQ and MaxQ the system randomly rejects incoming packets with a probability determined by a linear interpolation between 0% (at MinQ) to 100% (at MaxQ).

Hence, the node begins to randomly send a small number of congestion signals to some sources well before the full capacity of the buffer is reached. Subsequently, a wide range of variations on RED were produced in order to optimise a variety of behavioural properties of the network. However, the core mechanism remains the same – using randomly allocated “early” congestion signals to de-correlate resource usage requests.

3.2 Mapping RED Techniques to Grid Scheduling problems

As noted above, while multiple scheduler Grid environments may not have as many independent schedulers as IP networks have potential senders, there may still be strong analogies between the environments. In particular,

- As soon as multiple independent schedulers are present they will receive feedback from the same environment and hence the potential for oscillation driving feedback mechanism exists.
- Even if schedulers are not directly competitive, they have no strong incentive to cooperate with other schedulers as soon as they are not part of the same organisations, hence rational behaviour would lead each scheduler to be configured to gain maximum individual benefit from the Grid.

In the remainder of the paper we therefore pursue a simple mapping of IP network structures and the RED approach to the generic Grid scheduling problems described in Section 2:

- Processors P are analogue to IP network nodes.
- Jobs (tasks) are analogue to data packets.
- Job rejections are analogue to TCP/IP packet drop messages.
- Processors implement buffer management policies including RED.
- MinQ and MaxQ are set to fractions of the buffer length (see varied parameters in the next section).

The hypotheses to be tested are:

- That oscillations in resource allocation could also occur in multi-scheduler Grid environments.
- That the application of RED policies at resources may be a simple way in which to significantly damp such oscillations.

We further explore not only standard RED policies (termed *Static RED* from here onward) but also a variation of dynamic RED which further improves oscillation damping and resource usage.

4 Experimental Analysis

In order to test these hypotheses, this section covers a set of simulations (using the SMPL simulation engine [19]) of the performance of the random and shortest queue scheduling policies with and without RED. Of interest are how RED strategies perform on average compared to simple standard scheduling policies, and their effect on the local queues of the available resources.

4.1 Scheduling Strategies

The set of scheduling strategies studied were the following:

- Random without RED (R): Jobs ready to be executed are assigned randomly to the machines, regardless of the local queue load. This strategy represents the case of a pure dynamic method that has no information on the processors or the application.
- Shortest Queue without RED (SQ): Upon receiving a request for executing a job, each scheduler will use the information provided by a monitor system such as the EGEE Information Supermarket [20] to select the processors with the shortest queue. It should be noted that this monitoring information is accurate as sent by the processors to the schedulers, however there is a time delay before it arrives at each scheduler – the information is therefore gathered asynchronously, hence independent from the scheduling process and is updated every second.
- Random with RED (R-RED): This strategy consists of applying RED to the jobs when scheduled using the random strategy. The thresholds of RED are kept constant throughout all simulation runs.
- Shortest Queue with RED (SQ-RED): This strategy consists of applying RED to the jobs when scheduled using the SQ strategy. Once the machine with the shortest queue has been chosen to execute a job, it may reject that job depending on its queue occupancy. If the queue length is below MinQ of its capacity, the job will be accepted. Otherwise it may be rejected with a probability proportional to the queue occupancy, i.e. the closer the queue is to MaxQ, the higher the probability of rejecting the job.
- (All RED Schedulers) backoff mechanism: in addition to these behaviors, all schedulers working in RED environments implement a simple backoff policy in which, once rejected from a particular resource, they ignore this resource for a number of time steps before trying again (meanwhile they may send jobs, including the rejected job to other servers).

4.2 Simulation Framework

All described scheduling strategies are simulated in scenarios determined by the following parameters:

- *Number of Machines*: This represents the number of available processors on which the jobs are executed. We performed simulations using 20 homogeneous machines.
- *Number of Schedulers*: This represents the number of schedulers that receive requests for executing jobs, and the ones who schedule jobs using either the

random or the shortest queue policies. In our experiments the number of schedulers considered is 5.

- *Number of jobs*: Our experiments considered three types of jobs with execution times of 10, 20 and 30 time units, that are random distributed equitably in all simulation. Jobs are generated according to a exponential distribution with rate 5600 (resulting in approximately 5400 jobs).
- *Queue Length*: Each processor has an associated queue. In our experiments the maximum capacity of the queue was 30 jobs.
- *RED Thresholds*: RED will always accept jobs if the queue occupancy is up to 5 jobs. If the queue occupancy is bigger than this, RED will drop jobs with a probability proportional to the queue occupancy.
- *Backoff time*: the backoff time for all schedulers is set to $4 \times (\text{Queue length} - \text{MinQ})$.

We assumed that communication among the schedulers and the processors takes a fixed amount of time (10 time units for a job to reach the executing machine queue, and 5 time units for a rejection notification to reach the submitting scheduler). 8 time units are added to allow for the time required to clean the job from the system. The results show total job total execution time (from arrival at the scheduler to completion) and processor queue behaviour for the strategies considered.

4.3 Simulation Results – Oscillations and RED

In each of the figures the x-axis shows execution time in simulation units. The y-axis represents the queue occupancy in number of tasks or jobs for each of the machines. Fig. 1(a) shows the evolution of 4 of the 20 processor queues for the random scheduling policy (R), while Fig. 1(b) show the queue evolution for Random when RED is applied (R-RED). We show only 4 queues for the sake of clarity (the behaviour of these queues being typical).

Fig. 1. Comparing R scheduling with and without RED. The introduction of Static RED behavior (b) for the resources in the system hugely reduces the variable nature of resource usage in different systems queues – keeping average queue lengths much lower than without RED.

In the simulation runs shown, the striking feature is the high variability in queue length shown by the simple random strategy without RED as well as the clear mirroring behaviour of some queues of one another (Q3 decreasing and increasing in opposing oscillation to Q1 for example). The application of RED by contrast shows a radically different picture, with average queue occupancy more uniform across queues. The results shown are typical of many runs with variations of similar parameters and in terms of results that under R, a total (in the run shown) of 5387 jobs, the average turnaround time was 228.87 time units with a standard deviation on turnaround time of 137.15 time units. For R-RED, results for the run shown (with the same parameters) show for 5403 jobs and average turnaround time of 133.36 time units (almost half of the scenario with no RED) and a standard deviation in turn around time of 46.51 time units.

Fig. 2. Comparing SQ scheduling with and without RED. As with random, the introduction

of RED shows a radical change in system behavior.

As with random scheduling policies, shortest path allocation also shows a major change in results from non-RED to RED environments. In terms of raw numbers, on its 5399 jobs, SQ showed an average turnaround time of 220.6 and a standard deviation of 107.14 time units – both marginally improved in comparison with the R strategy. With RED however, average turnaround times and standard deviations drop to 132.73 and 52.47 time units respectively, also in each case almost halving the times involved.

While only one plot is shown, many experiments with different random number seeds and varied parameters were executed – all of these showed the same underlying dynamics (examples include significant variations in RED MinQ / MaxQ parameters, increases in job execution time to double their current value with reduction in arrival rate, as well as variations in the number of schedulers [10-100 and 15-300 respectively]).

In particular, for both sets of results shown here (and for all results more generally), key significant features include:

- The average job turnaround time drops dramatically – to nearly half of the values seen without RED. On closer analysis, in particular the longest waiting jobs under both R-RED and SQ-RED are processed much more quickly than under R and SQ, bringing a very large reduction in the standard deviation in job execution time.
- RED's effect is to smooth off the peaks of oscillations as they build and avoid jobs regularly forming unnecessarily long queues at resources which due to delayed load information appear to be attractive to all schedulers.
- This positive result is seen despite the fact that results include extra costs incurred by some jobs as they are resubmitted.

The results (the examples shown and more generally) also show that SQ performs only marginally better than R without RED and performance is essentially identical when using RED. While this appears to be counter-intuitive (since SQ would appear to be the “smarter” scheduling strategy), this effect reflects a cancelling out of the positive effects of queue selection by the negative impact of oscillations brought on by unintended synchronisation among job submissions by SQ. The synchronisations and peaks seen under R appear to be a clear indication of the presence of the types of dynamics typically predicted for balls and bins problems [20 Raab98], in which random assignments in fact often cause large differences (clustering) in the resources items are assigned to.

The conditions shown here are particularly susceptible to oscillations since processor usage information is somewhat time delayed (10 time units) meaning that the actions of other schedulers are hidden for a short amount of time to other schedulers and actions take time to have an effect (hence a job sent at time t , does not become visible in the queue until $t+10$). While these are arguably not extreme parameters:

- Schedulers all poll at the same rate – giving them the same processor picture at each decision step.
- Schedulers all use the same assignment policy (shortest queue).

To some extent these factors therefore accentuate the possibility of the emergence of oscillations. However, it is also clear that even if some conditions are softened oscillations would still be expected:

- Modelling different polling rates for information updates for different schedulers mean less synchronized world models. However, these would still ultimately be (time-delayed) models of the same reality – hence ultimately such scenarios also show synchronisation.
- While schedulers may use different heuristics for scheduling, shortest queue is the rational individual choice – making it unlikely that an individual scheduler would choose another policy.

The impact of RED on the random case is illustrative, since even here the extra early rejection factor manages to cut out the build up of long queues. The effect however also points to potential problems in using simple RED in very highly loaded Grid environments since the rejection policy applied when many buffers are near capacity is likely to cause a large number of rejections and re-submission. This

pathology is well known for simple RED and a number of other techniques (e.g. see Section 5) are available which go beyond RED to counter this type of effect. It can also be observed that RED in its static form only begins to have a significant effect once system load enters a “critical zone” in which queue occupation is between the RED $MinQ$ and $MaxQ$ thresholds – limiting its effectiveness, in for example, preventing congestion in the first place.

4.4 Dynamic RED

In order to address some of the limitations seen in static RED, further experiments were carried out in which the thresholds which govern RED behaviour were adapted dynamically to system load. In particular, Dynamic RED thresholds are calculated in the following way:

$$\begin{aligned} MaxQ &= (Queue\ Length - Current\ Queue) \cdot \alpha \\ MinQ &= MaxQ \cdot \alpha \end{aligned} \quad (0)$$

where, as before, $MaxQ$ is the maximum threshold, $MinQ$ is the minimum threshold, α is a value between 0 and 1, $Queue\ Length$ the queue maximum capacity and $Current\ Queue$ the current queue occupancy. Depending on system-load, the administrator can set the α value in order to adjust/tune system performance. However, in all experiments reported we set α to 0.2. The effect of the changes to RED is to create a mechanism which tightens and loosens control on a resource based on load. On the basis of this mechanism two more scheduling cases are studied:

- Random with Dynamic RED (R-DRED): which consists of applying Dynamic RED to the jobs when scheduled using the random strategy.
- Shortest Queue with Dynamic RED (SQ-DRED): applying Dynamic RED to the jobs when scheduled using the SQ strategy.

Results for experiments using the same configurations as used in the previous section are shown in Fig. 3.

Fig. 3. Showing scheduling cases random with dynamic RED (left) and shortest queue with Dynamic RED (right). Results show a further improvement on the static RED case – further reducing round trip times and standard deviations.

Using Dynamic RED, results improve further (though not dramatically since the major step shown even with static RED is the cut through the majority of delays caused by oscillations):

- For R-DRED, the run shown of 5467 jobs has an average turnaround time of 130.62 and a standard deviation of 45.12 – not a significant advance on R-RED.
- For SQ-DRED, the run shown of 5455 jobs has an average 123.96 and a standard deviation of 48.94, representing a small but significant improvement for SQ (which can be seen consistently over other runs with similar parameters) which also makes SQ-RED marginally better than R-RED and R-DRED in performance.

5 Comparative Analysis of Results

In order to illustrate the difference in operation between the techniques in question, Fig. 4 compares the average number of jobs per resource currently waiting in the system under gradually increasing job arrival rates. The graph for each scheduler type and resource policy (no RED, static RED and dynamic RED) hence shows the point at which a particular system becomes saturated by latent jobs in the system. Since jobs could be queued at a resource (the maximum buffer length at each is 30) and at the scheduler (if rejected from a resource), the graph sums both figures to create the average.

Load is generated according to the same exponential distribution used in the previous experiments but the number of jobs per second is gradually increased from

0.5 jobs per time unit (50% of system theoretical maximum throughput) through 1 job per time unit (100% of theoretical maximum throughput) at 10,000 time units to 1.1 jobs per time unit (110% of capacity) after 12,000 time units.

Fig. 4. Comparative behavior under increasing system load for Random (left) and Shortest Queue scheduling. With both R and SQ approaches, RED approaches keep residual jobs in the system significantly lower for longer – keeping more jobs flowing through the system.

The results of this comparison show:

- A clear difference between RED and non-RED strategies at high levels of throughput (from 5000 time units / job arrival rate at 75% of theoretical maximum) in the R case and even at low levels of throughput (50% of theoretical maximum) in the SQ case.
- Dynamic RED performs marginally better than static RED throughout.
- SQ also generally has slightly worse performance at lower levels of resource occupancy than R – highlighting the impact of local, small scale oscillations creating unnecessary, short lived synchronisations in the network.
- At higher job arrival rates however, non-RED systems benefit from shortest queue which keeps static load in the system lower for longer than the Random scheduler.

In terms of concrete figures for these simulation runs:

- Random without RED showed an average turnaround time of 101.5092, with standard deviation of 80.8737. With static RED this dropped to an average turnaround of 82.3545 and a standard deviation of 45.6551. With dynamic RED, figures drop to 72.1976 and 36.5258 respectively.
- Shortest Queue results follow the same pattern, with non-RED average turnaround time at 153.4731 (deviation 88.1687), static RED dropping this to 109.7881 and

53.6473 and dynamic RED to average turnaround time of 90.2004 and standard deviation of 48.0186.

6 Discussion and Related Work

Advances in Grid scheduling techniques have been very significant in recent years. In particular, resource controllers such as NQE [1], Utopia [2], Load leveller [3] and Condor [4] work very effectively in environments where accurate / up to date load information can be obtained and the scheduler is the primary actor allocating resources.

However, as seen in the results presented here, it is not clear if these successes can be translated into use for multi-scheduler environments in a straightforward way. Although experiments are preliminary and some of the phenomena are no doubt dependent on the particular parameters chosen a clear pattern is seen. Furthermore, as the number of schedulers increases, if all schedulers use the same metric for allocating jobs, then oscillations are increasingly likely to occur.

Such phenomena could therefore present a major problem in future multi-scheduler Grid environments.

Work such as that in [22] and [8] (on the coordination of multiple schedulers), [9] (on agent based distributed scheduling) and approaches such as Market Based Scheduling [10] are beginning to address this problem and understand:

- Environments in which independent actors compete for resources.
- Act over the same resources.

The methods presented in this paper argue that, in addition to such approaches a further powerful tool may include the adaptation RED and other similar buffer management techniques may provide a useful tool for tackling such pathologies given their widespread usage and adaptation (structurally similar TCP/IP network problems). The approach is different (and complimentary) to those which, for example, rely on the hierarchical organisation of schedulers to avoid conflict. Effectively, a hierarchical or other a priori division of resources provides a useful means to avoid conflict, without however allowing flexible subsequent usage of all resources since some regions may be under-used while others are overloaded. Re-planning approaches to change hierarchical dependencies can be useful to change

provisioning. However, these still do not allow real time access to all resources for independent actors.

Despite the early promise shown here however, we caution that the results are preliminary and a wide range of further work remains. In particular:

- Static RED policies in IP networks, while making a significant impact are also not a complete solution to the oscillation problems present. This has led to a wide variety of RED-variations specifically targeted at solving some of RED's deficiencies (examples include [18], [23], [13] and other techniques such as [14]).
- While the mapping between IP networks and Grid scheduling is natural, a wide variety of parameters assumed fixed here (such as job length, variations in communication times etc.) remain to be considered.
- IP network traffic allocation tends to be “open loop” such that users generate traffic independently of one another (hence loads can be arbitrary), however closed loop models (where the success / timing of previous jobs and the state of the system in general - throttling the load in the system) might also apply to Grid scheduling environments.
- In particular, applying routing metaphors to jobs which require multiple different processors in sequence appears to be a rich and challenging area.
- If the link between Grid resource scheduling and TCP/IP network management can be made, a huge body of existing research would be opened up in areas such as security, denial of service, performance tuning and other areas.

7 Conclusions

Whilst production Grid environments with many multiple independent scheduling processes may still be way time off, they are likely to grow significantly in importance. In this paper, we argue that such environments may suffer from similar oscillation problems to IP network traffic management scenarios and that, further, approaches based on Random Early Detection policies may be a strong candidate for mitigating some of these pathologies.

While results are preliminary they arguably show promise:

- The oscillations seen are potentially an underlying phenomenon which will have a significant impact on the design and use of multi-scheduler systems in large Grids.

- RED shows significant promise for tackling such oscillations and further, if validated, has a number of attractive properties for deployment in Grids including no need for centralised management, deployment at individual processor sites only and a wide range of flexibility on the precise drop strategy.
- IP network scheduling and routing techniques more generally may hence be a *very significant source of inspiration* for scheduling challenges in next generation Grid environments.

Future work includes exploring a wider range of simulation parameters in order to validate basic results and exploring a range of the challenges discussed in Section 6.

References

1. Cray Research Inc.: Introducing NQE v3.3, IN-2153. Research Report (1993)
2. Zhou, S., Zheng, X., Wang, J. and Delisle, P.: Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems. Software - Practice and Experience, Vol. 23, No. 12 (1993) 1305-1336
3. IBM, Load Leveler; User's Guide, IBM (1993)
4. Thain, D., Tannenbaum, T. and Livny M.: Distributed Computing in Practice: The Condor Experience. Concurrency and Computation, Practice and Experience, 2004.
5. The Gridbus project; <http://www.gridbus.org>
6. Berman, F., et al.: The GrADS Project: Software Support for High-Level Grid Application Development. In: International Journal of High Performance Computing Applications, Vol. 15, No. 4, (2001) 327-344
7. Berman, F. and Wolski, R.: The AppLeS Project: A Status Report. 8th NEC Research Symposium. Berlin, Germany (1997)
8. He, L., Jarvis, S., Spooner, D., Chen, X. and Nudd G.: Dynamic Scheduling of Parallel Jobs with QoS Demands in Multiclusters and Grids. In: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04), DC, US (2004) 402 – 409
9. Cao, J., et. al.: ARMS: An Agent Based Resource Management System for Grid Computing. Scientific Programming, Vol. 10, No. 2 (2002) 135-148
10. Wolski, R., Plank, J. S., Brevik, J. and Bryan, T.: Analyzing Market-Based Resource Allocation Strategies for the Computational Grid. The International Journal of High Performance Computing Applications, Vol. 15, No. 3 (2001) 258-281
11. Floyd, S. and Jacobson, V.: Random Early Detection gateways for Congestion Avoidance. ACM Transactions on Networking, Vol. 1, No. 4 (1993) 397-413
12. Braden, B., et al., Recommendations on Queue Management and Congestion Avoidance in the Internet. RFC IETF 2309 (1998); <http://www.ietf.org/rfc/rfc2309.txt>
13. Kelly, F., Maulloo, A. and Tan D.: Rate control for communication networks: Shadow prices, proportional fairness and stability. Journal of the Operational Research Society, Vol. 49 (1998) 237-252

14. Low, S. and Lapsley, D.: Optimization flow control--I: basic algorithm and convergence. IEEE/ACM Transactions on Networking, Vol. 7 (1999) 861-874
15. Postel, J. (ed.): Transmission Control Protocol (TCP), IETF RFC 793 (1981); <http://www.ibiblio.org/pub/docs/rfc/rfc793.txt>
16. Hedrick, C.: Routing Information Protocol(RIP), IETF RFC 1058 (1988); <http://www.ietf.org/rfc/rfc1058.txt>
17. Partridge, C.: Isochronous Applications Do Not Require Jitter-Controlled Networks. IETF RFC 1257 (1991); <http://tools.ietf.org/html/rfc1257>
18. Hollot, C. V., Misra, V., Towsley, D. and Gong, W.: A Control Theoretic Analysis of RED. In: Proceedings of INFOCOM (2001) 1510-1519
19. MacDougall, M.: Simulating Computer Systems, Techniques and Tools. The MIT Press (1992)
20. EGEE Information Supermarket; <http://jra1mw.cvs.cern.ch:8180/cgi-bin/jra1mw.cgi/org.glite.wms.ism/>
21. Raab, M. and Steger, A.: Balls into Bins - A Simple and Tight Analysis. Lecture Notes in Computer Science, Vol. 1518. Springer Berlin Heidelberg (1998) 159-170.
22. Pazel, D. P. et. al.: Neptune: A dynamic Resource Allocation and Planning System for a Cluster Computing Utility. In: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid. IEEE Computer Society. Washington, DC, USA (2002)
23. Floyd, S., Gummadi, R. and Shenker, S.: Adaptive RED: An Algorithm for Increasing the Robustness of RED. Technical Report, under submission (2001)