Current methods for negotiating firewalls for the Condor® system

Bruce Beckles[1], Se-Chang Son[2] and John Kewley[3]

[1]University of Cambridge Computing Service, New Museums Site, Pembroke Street, Cambridge CB2 3QH, UK
[2]Computer Sciences Department, University of Wisconsin, 1210 W. Dayton Street, Madison, WI 53706-1685, USA
[3]CCLRC Daresbury Laboratory, Daresbury, Warrington, Cheshire WA4 4AD, UK

Abstract

The Condor® system is a widely used "specialized workload management system for compute-intensive jobs" [1] which is increasingly being used in UK academic environments to utilise the so-called "idle time" of workstations. Due to Condor's pattern of network communication there are a number of issues that arise when Condor is deployed across firewall or private network boundaries. In this paper we describe and analyse these issues, and outline the characteristics that we believe a general solution to these issues would have. We briefly describe some currently available solutions and workarounds, and then identify the most promising direction for future developments.

1. Introduction

The Condor® system [1] is a batch queuing system that is particularly suited to harnessing the so-called "idle time" on workstations and clusters and is frequently used for "high throughput" or "compute-intensive" jobs. Condor is widely used in UK academic environments, both to maximise the return on investment (ROI) of existing computing infrastructure and to allow researchers inexpensive access to resources capable of supporting high throughput computing (HTC).

Unfortunately, Condor was designed to run in a network environment which is both "symmetric" (i.e. one in which any machine can initiate a connection to any other machine), and in which there are no restrictions on types of network traffic (e.g. firewalls blocking UDP). In the modern computing environment such an "open" network environment is increasingly rare. It is thus the case that it can be quite difficult to deploy Condor in many current network environments due to the presence of firewalls, private networks (i.e. networks of machines with IP addresses in the range specified by RFC 1918 [2]) and other circumstances that "break the symmetry" of the network (see [3] for a fuller discussion).

In attempting to address the issues that arise when Condor is deployed across a firewall or private network boundary it is important to remember and respect the purpose of the firewall or private network. Often this is to provide a layer of security for the machines behind the firewall or on the private network and it is therefore vital that this security layer is not compromised by attempts to deploy Condor across that security boundary.

In this paper we describe Condor's pattern of network communication and explain why this pattern of communication is so inimical to firewalls and private networks. We then list the requirements that would be desirable in a general solution to these problems, and finally we review some of the solutions / techniques which have been developed to address or mitigate these problems.

2. Condor's pattern of network communication

As the Condor system continues to evolve it is likely that its pattern of network communication will change. The details given in this section are intended to cover primarily the Condor 6.6 series up to version 6.6.10, and secondarily the Condor 6.7 series, up to version 6.7.8, which are the current versions at the time of writing. In addition, it should be borne in mind that the Condor system is a complex one that allows for many diverse patterns of deployment, and only some of the most common patterns of deployment are covered here – for instance, we do not cover the scenario in which the functions of the central manager (see Section 2.1) are split between different machines, nor do we discuss CondorView servers.

2.1. Machine Roles

To understand Condor's pattern of network communication it is necessary to understand something of the structure of the Condor system, and, in particular the different roles which a machine running Condor may have. For more details, see [4], but, in summary, the most significant roles are as follows:

- Submit nodes: These are machines that submit jobs to the Condor pool.
- Execute nodes: These are machines in the Condor pool that execute users' jobs.
- Central manager: This is the machine that monitors all the other nodes and "matches" jobs to execute nodes.

There must be at least one machine in each of the above roles in order for the Condor pool to function. There can be only one machine actively taking the role of the central manager, although in the later releases of the Condor 6.7 series, there may be other nominated machines (known as idle central mangers) that may act as the central manager should the active central manager machine fail. In addition to the above roles, there is another (optional) machine role that is often found in a Condor pool, namely:

- Checkpoint server: This is a machine that stores checkpoints of jobs submitted to the pool, for those types of jobs that support checkpointing. In the absence of a checkpoint server, the submit node from which the job was submitted will be used instead.

Upon first encountering the Condor system, a common misconception is that the system works as follows: submit nodes submit jobs to the central manager, which then sends them to an execute node, receives the results from the execute node, and then sends them back to the submit node from which the job originated. This is completely incorrect: what actually happens is that the central manager receives a copy of the job's characteristics, which it matches against execute nodes' characteristics. When it has made a match it then contacts the submit and execute nodes in question, which thereafter communicate directly with each other; the central manager is then no longer involved. (This is also how jobs are handled when separate Condor pools are connected via Condor's flocking mechanism.)

2.2. Direction of network communication

In many firewall configurations, especially for stateful firewalls and devices that enable network communication across the boundary of private networks, the direction of network communication is extremely important. Typically there will be one set of rules for *inbound connections* (i.e. those connections initiated by machines *outside* the private network or firewall boundary) and a different set of rules for *outbound connections* (i.e. those connections initiated by machines *inside* the private network or firewall boundary). In particular, a "deny all inbound connections; permit all outbound connections" (or close variant) is a particularly common policy with stateful firewalls and gateways to private networks.

In the Condor system, most machines need to be able to both initiate and receive connections from most other machines that are part of the same Condor pool, at least in the most common configurations of the pool – this pattern of network communication is known as "many-to-many". Table 1 gives details of which machine roles initiate connections to which other machine roles, and which network protocols (TCP, UDP or both; see Section 2.4) are used (note that in this table initiators and recipients are presumed to be *distinct* machines). This pattern of communication is incompatible with many common firewall policies, which are usually designed with a "one-to-many" (or possibly a "few-to-many") pattern of network communication in mind.

Recipient: / *Initiator:*	CM	Ckpt	S	E
Central Manager (CM)	*N/A*	×	TCP	TCP UDP
Checkpoint Server (Ckpt)	TCP UDP	×	×	×
Submit (S)	TCP UDP	TCP	×	TCP UDP
Execute (E)	TCP UDP	TCP	TCP	×

Table 1: Initiators and recipients of network connections (and protocols used) in the Condor system. Note that this Table does not take into account the high availability daemon (available in Condor 6.7.6 and later).

2.3. Network port usage

In general, network port usage by an application can be divided into two categories: *static* ports and *dynamic* (or *ephemeral*) ports. Static ports are ports that are always used by a particular instance of an application throughout its lifetime, and are usually known in advance rather than 'randomly' chosen at run-time (e.g.

port 22 for SSH servers). Once set, an application will always use a particular static port for particular functions. A dynamic or ephemeral port is one that is chosen (often 'randomly') from a particular port range when the application needs to use a port. Once the application has finished using that port, it will close it. When it needs to use another port, another port from the given port range will be chosen (which may or may not be the same as the previous port).

Condor uses both static and dynamic ports. Normally, the central manager uses two static ports (by default 9614 and 9618) – as of Condor 6.7.5, this can be configured to be only a single static port (by default 9618) – which can be changed in Condor's configuration file. If the high availability daemon is being used (Condor 6.7.6 and later) then an additional static port (configured in a configuration file) is used by the active central manager and by the idle central manager(s). Checkpoint servers use four static ports (5651, 5652, 5653 and 5654) and these cannot currently be changed.

In addition, all machines use a number of dynamic ports. The range from which these are drawn is, by default, all valid port numbers above 1023, but this range can be changed in the Condor configuration file to any sub-range of the default range. If this range is too small, then the Condor daemons will not function properly: the minimum acceptable size of this range depends on the role of the machine in question and a number of other factors (see [5]). For example, on submit machines, the size of this range may limit the number of jobs that a submit machine can run simultaneously – thus this range may need to be quite large.

One factor not mentioned in [5] that also affects the acceptable size of this range is that under many circumstances the Condor daemons will be unable to reuse the dynamic ports in this range immediately. This may mean that the size of the range needs to be increased above the minimum size given in [5] if Condor is to function properly.

Generally firewall administrators are most happy with services that only use a few static network ports for inbound connections. Unfortunately, this will often not be the case in a Condor pool, and the range of dynamic ports that are used may be very large, requiring the firewall administrator to open a large number of holes in their firewall.

2.4. Network protocols

For performance reasons, much communication between machines in a Condor pool uses the UDP network protocol, although there is significant use of the TCP network protocol as well. Machines will periodically send status messages to other machines in the pool and this normally is done over UDP. Starting in the Condor 6.5 series, it has been possible to configure much (but not all) of this communication to use the TCP network protocol instead, although doing this introduces performance overheads and means that some of the Condor daemons require additional memory.

This is an issue for firewalls because the default configuration of many firewalls is to block UDP, and security considerations mean many firewall administrators are reluctant to allow UDP across their firewall. In addition, network devices and TCP/IP stacks process UDP packets differently to TCP packets, and, as UDP is by design unreliable and so only infrequently used for key network communication without an additional transport layer, many network devices and TCP/IP stacks do not handle UDP as well as they ought. Thus networks and operating systems which have been perfectly adequate for applications that mainly or solely use TCP may prove inadequate for applications like Condor that make extensive use of UDP for important messages without implementing an additional transport layer on top of the UDP protocol.

2.5. Other issues

There are a number of other issues concerning Condor's pattern of network communication and firewalls / private networks that may not be immediately apparent from the preceding sections, or that have not yet been mentioned. Some of these are listed below:

- *Administrative overhead:* As the number of machines on either side of the firewall or private network boundary increases, the administrative load on the firewall or network administrator may rapidly become unacceptable. In addition, the necessity of involving the firewall or network administrator may make expanding the Condor pool an administratively burdensome process.
- *Personal firewalls:* A personal firewall is a firewall that runs on an individual machine where that individual machine is the only machine behind the firewall boundary. In an environment where personal firewalls are deployed (and such environments are increasingly common) the personal firewall on each machine will need to be adjusted if the machine is to be part of the Condor pool and may also need to be adjusted every time

a new machine is added to the pool. The administrative overhead in managing this may rapidly become extremely burdensome.

- *Condor does not handle certain network problems gracefully:* Because Condor was designed to be run in a symmetric network environment, it does not handle many types of network failure gracefully, simply because the possibility of these types of failures was never considered in its design.
  For example, if the central manager can communicate with an execute node, but a submit node cannot, jobs from that submit node may still be matched to that execute node. If this happens the submit node will become "stuck" and the smooth handling of other jobs from that node may be affected. Also, Condor will not realise that there is a problem with communication between the particular submit and execute nodes, and may keep attempting to run jobs from that particular submit node on the execute node that is inaccessible to it.
- *Documentation:* Unfortunately the official Condor documentation regarding Condor's pattern of network communication is somewhat sparse and certainly incomplete. In addition, there is outdated or inaccurate documentation by other individuals or organisations in circulation. As this area is quite complex, there is an urgent need for accurate comprehensive documentation of Condor's network behaviour.
- *Bugs in Condor:* Like any complex piece of software, Condor will inevitably have bugs, and some of these have been known to affect its performance in the presence of firewalls. For example, prior to Condor 6.6.8 and Condor 6.7.3, the `SO_KEEPALIVE` option on network sockets was not set under certain circumstances, and this meant that firewalls which terminated apparently inactive connections after a certain period of time might erroneously terminate Condor's network connections between submit and execute nodes, with catastrophic results for the job running on the execute node.
  At the time of writing, there are still issues involving machines that "disappear" from the Condor pool, although the machine in question is actually functioning fine and has not suffered a loss of network connectivity. There also have been problems with Condor failing to automatically negotiate the Windows® Firewall under Windows® XP Service Pack 2 and Windows® Server 2003 Service Pack 1, although these are believed to have been fixed in Condor 6.6.10 (and the forthcoming Condor 6.7.9).

2.6. Summary

Table 2 presents a summary of the main issues identified in this section.

Issues Identified
"Many-to-many" / bi-directional pattern of communication
Uses large range of dynamic ports
Uses both TCP and UDP protocols
High administrative overhead for firewall administrators
Not designed to be "personal firewall friendly"
Does not fail gracefully in the presence of firewalls or private networks
Inadequate documentation
Unresolved bugs relating to network communication

Table 2: Issues identified

3. Identified requirements

Our analysis of Condor's pattern of network communication, combined with discussions with Condor administrators and firewall administrators in the UK and abroad, as well as our own experiences in attempting to resolve some of these issues, has led us to identify the following requirements as highly desirable and / or essential for any solution (or partial solution) to the problems highlighted in Section 2:

- *Respect the security boundary:* The security boundary established by the firewall or private network must be respected, and exposure to external attack must not be increased by the solution.
- *Reduce administrative overhead:* The administrative overheads of the firewall administrator(s) and Condor administrator(s) must be reduced (or at worst not increased) by the solution.
- *Minimal impact on performance of network "choke points":* The solution must have minimal impact on the performance of existing network "choke points" such as firewalls and gateways to private networks. In practice this may mean reducing Condor's pattern of network connection from "many-to-many" to "few-to-many" or better ("one-to-many", "one-to-few", etc).
- *Enable traversal of firewall and private network boundaries:* A desirable feature in a general solution to the issues described in Section 2 would probably allow the traversal of firewall and private network boundaries

for Condor traffic. However, this must be balanced against the risks to which such traversal might expose a site.

- *Allow incremental implementation:* It must be possible for the solution to be incrementally implemented across the machines concerned. In particular, the situation where only some machines are "aware" of the solution and make active use of it needs to be catered for.
- *Scalability:* The solution needs to be scalable, as large Condor pools may contain thousands of machines, and often separate Condor pools are joined together across network security boundaries, and such flocks of Condor pools may comprise many thousand of machines.
- *Robustness:* The solution should be robust in the face of network congestion.
- *Gracefulness:* The solution should fail and recover gracefully from network problems – in particular it should handle the situation discussed in Section 2.5 where some, but not all, of the machines in a pool can communicate with a particular node.
- *Integration into Condor's security framework:* If the solution is part of the Condor system it must be fully integrated with Condor's security framework (authorisation, etc).
- *Logging:* The solution should provide comprehensive logging facilities.
- *Documentation:* The solution must be clearly and comprehensively documented.

4. Current solutions

There are a number of current solutions or partial solutions that attempt to address the issues described in Section 2. In this section we briefly describe some of them and then see whether they meet the requirements listed in Section 3. These solutions can be divided into three categories: "*mitigation*" (mitigating the effects of firewalls, etc), "*altering the pattern of network communication*" (e.g. reducing it to "one-to-many") and "*firewall/NAT traversal*" (traversing the security boundary).

4.1. CCLRC's "Firewall Mirroring" (FM)

One of the authors (John Kewley) has developed a method of configuring the submit and execute nodes in his Condor pool so that jobs will not be submitted to machines which cannot run them because communication is not possible between the submit and execute nodes. This is achieved by duplicating part of the firewall's configuration in the ClassAd of each execute node, and then modifying the Requirements of the Condor job so that it will only match with execute nodes that can communicate with its submit node. For details of this procedure, see [6].

This method is particularly useful with personal firewalls that are not centrally managed. Although it does not help in traversing firewall or private network boundaries, it mitigates the effect of such boundaries, since it allows the pool to continue functioning even when firewall configurations are not up-to-date, or the pool is partitioned by firewalls and/or private networks. However there is an administrative overhead associated with this solution, which means that it is probably unsuitable for large pools.

An example scenario for this solution would be a small pool of machines, each with their own personal firewalls (that are not centrally managed), such as the Condor pool at CCLRC [7].

4.2. Using centralised submit nodes (CS)

Another way of addressing many of the problems described in Section 2 is to reduce the number of submit nodes, and to place all these nodes in the same part of the network. Centralising submit nodes in this way reduces the pattern of network communication from "many-to-many" to "few-to-many" (or even "one-to-many"). Although any firewalls still have to be configured to allow traffic from these submit nodes through, the small number of submit nodes means that the administrative overhead is considerably lowered.

In addition, the impact on the firewall's performance is likely to be small, and this can probably be made even lower if the centralised nodes have IP addresses that form a contiguous range that can be expressed in the Classless Inter-Domain Routing (CIDR) [8] notation. However, centralising submit nodes in this manner may reduce the availability of the Condor pool as such a pool architecture may well have a single point of failure. In addition the reduction in the number of submit nodes must be balanced against the size and needs of the pool. If the number of submit nodes is too small, then the number of jobs that can be simultaneously run may be significantly less than the number of execute nodes in the pool. It may also be the case that this solution cannot handle more than one private network.

An example scenario where such a solution might be appropriate would be a University "campus grid" where all the compute nodes were centrally managed, such as the University of Cambridge Computing Service's Condor deployment [9].

4.3. Remote job submission/Condor-C (C-C)

Another way to reduce the pattern of network communication from "many-to-many" to "few-to-many" (or even "one-to-many") is to make use of Condor-C [10] or Condor's remote job submission feature. Condor-C is a new feature, added in Condor 6.7.3, that allows the job submission queue on one submit node to be moved to another submit node and "scales gracefully when compared with Condor's flocking mechanism" [10]. Condor-C maintains only a single network connection between the submit node which originally held the job queue and the submit node to which the queue has been moved. Condor's remote job submission feature allows a job to be submitted from a machine to a remote submit node.

Either of these features could be used to reduce Condor's pattern of network communication by careful construction of the Condor pool. There are a number of possible architectures, such as the following:

- Condor traffic across the firewall or private network boundary could be restricted to a small number of "internal" submit nodes behind each firewall (or on the border of each private network). Machines outside these security boundaries would then use Condor-C or Condor's remote job submission feature to submit jobs to the designated submit nodes inside (or on the border of) the security boundary. This would reduce Condor's pattern of network communication at the security boundary to "many-to-few" or "many-to-one".
- A small number of (probably centralised) "external" submit nodes that are outside any firewall or on the border of a private network could be allowed to send Condor traffic across the firewall or private network boundaries. Any other machine that wished to submit jobs would then use Condor-C or remote job submission to submit to these designated machines. This would reduce the pattern of network communication at the security boundary to "few-to-many" or "one-to-many".
- Both of the above architectures could be combined. This would mean that arbitrary machines outside a relevant security boundary would use Condor-C or remote job submission to submit jobs to the designated "external" submit nodes, which would then use Condor-C or remote job submission to re-submit those jobs to the "internal" submit nodes (behind the firewall or on the border of the private network). This would reduce the pattern of network communication across the security boundary to "few-to-few" or better ("one-to-few", etc.).

There are, however, a number of disadvantages with such architectures. There are scalability issues similar to those for the architecture described in Section 4.2. Also, Condor's remote job submission is poorly documented, does not scale well and there are security implications in using it. Whilst Condor-C does not suffer from any of the aforementioned disadvantages of remote job submission, it is still a very new feature that is currently only available in a development release of Condor.

In addition, the authors are unaware of any sites that are currently using any of the architectures described above or anything similar. This is likely to change as Condor-C matures and is accepted and used by the Condor user community, but at present this solution is probably best regarded as "experimental".

A generalised scenario in which such an architecture might be appropriate is one in which there are a number of separate Condor pools, each entirely behind a firewall or on a private network.

4.4. Generic Connection Brokering (GCB)

Currently being added to the development release of Condor is a firewall/NAT traversal technique known as Generic Connection Brokering (GCB), which was developed by one of the authors (Se-Chang Son). GCB allows the direction of the network connection to be independent of which machine initiated the connection (thus allowing traffic across security boundaries where traffic is restricted in one direction) and also incorporates a relay mechanism (allowing, for instance, nodes on two disjoint private networks to communicate with each other).

GCB is intended to be scalable and transparent to the application making use of it (it is implemented as a library directly beneath the application layer). Site administrators merely need to employ a GCB agent or broker in the appropriate part of their network and GCB then transparently enables communication across the firewall or private network boundaries (i.e. an appropriately placed GCB agent or broker could enable the traversal of multiple firewall and/or private network boundaries). For technical details and a more detailed discussion of GCB see [11] and [3].

GCB enables firewall/NAT traversal and effectively reduces the pattern of Condor's network communication to "one-to-many" on the firewall or private network boundary – the GCB agent or broker acting as a single "choke point". There are however a number of issues to be considered when deploying GCB.

As GCB is not yet integrated into Condor's security framework, it will allow any GCB-enabled application to traverse the firewall or private network boundary, and this has serious security implications. In addition, although designed to be scalable, there are scalability issues with GCB (see [3] for details), the limiting factors being the resource limits imposed by the operating system on the GCB process. As GCB is still not part of any Condor release (although it is expected to be added to the Condor 6.7 series soon), it must be regarded as an "experimental" solution.

A scenario in which GCB might be usefully deployed is where Condor needs to be deployed across a firewall or private network boundary in a manner transparent to the firewall or private network, for instance where the firewall administrator is unwilling or unable to open any additional holes in the firewall for Condor.

4.5. Dynamic Port Forwarding (DPF)

Dynamic Port Forwarding is a firewall/NAT traversal solution developed by one of the authors (Se-Chang Son, a member of the Condor Team). It is implemented through an add-on to the firewall (and so only supports certain firewalls, currently only firewalls based on Linux netfilter [12]). The basic idea is that when a DPF-enabled application wishes to traverse a DPF-enabled firewall it sends a request to the firewall and DPF then opens a hole in the firewall for the application. When the application has finished communicating across the firewall, DPF closes the hole it opened in the firewall. For further details of DPF and a discussion of its performance see [13] and [3].

Whilst DPF is highly scalable (see [3]), it does not reduce the pattern of network communication of any application that uses it, and there are a number of issues that must be considered before deploying it. Any DPF-enabled application will be able to get a DPF-enabled firewall to open holes for it, and this has serious security implications. DPF is not yet widely used and so should be considered an "experimental" solution.

A scenario in which DPF might be used is where a firewall administrator either cannot or does not wish to undertake the administrative burden of re-configuring their firewall for a trusted DPF-enabled application, or where they wish to minimise the exposure of the machines behind their firewall through the holes opened for a trusted DPF-enabled application.

4.6. Assesment of solutions according to requirements

Table 3 shows whether the requirements given in Section 3 are met by each of the five solutions described above.

	FM	CS	C-C	GCB	DPF
Respect security boundary	✓	✓	✓	×	×
Reduce administrative overhead	×	✓	✓	✓	✓
Minimal impact on firewall performance	✓	✓	✓	✓	✓
NAT/firewall traversal	×	×	×	✓	✓
Incremental implementation	✓	×	1	✓	✓
Scalability	×	×	×	2	✓
Robustness	×	×	×	×	×
Gracefulness	✓	×	×	✓	✓
Integration into Condor security framework	×	×	×	3	×
Logging	N/A	N/A	N/A	✓	✓
Documentation	4	×	×	4	4

Table 3: Whether solutions meet requirements

Notes:
1. Incremental implementation is possible to a certain extent by the gradual inclusion of those Condor pools entirely behind a security boundary.
2. Some scalability issues, see [3].
3. Integration planned.
4. Some documentation available.

5. The Way Forward

As can be seen from Table 3, none of these solutions are entirely satisfactory, although depending on the scenario, some of them may offer considerable advantages over the current situation in which *all* the issues described in Section 2 are unresolved. It is clear that much work remains to be done in the area of security: at present either solutions do not enable firewall/NAT traversal or they do enable this with potentially disastrous consequences for the security of the site.

In addition, some of the requirements given in Section 2, such as robustness in the face of network congestion and failing gracefully when firewalls or private networks are present, would be best addressed by changes within the Condor system itself, rather than attempting to work around these problems with the solutions described here.

It may be the case that, contrary to the spirit which the Condor system attempts to embody, the best solution in many cases is to carefully plan the deployment of Condor pools rather than allowing them to grow in an ad-hoc fashion. However, since there are clearly instances when firewall/NAT traversal of some description is the only practical solution, further development is needed in this area, particularly to address the security concerns.

6. Conclusion

The Condor system is an extremely useful batch queuing system that enables high throughput computing. Unfortunately, it is the case that it has not been designed to peacefully co-exist with firewalls or to support private networks. A number of techniques have been developed that attempt to address some of the issues that arise as a result of Condor's assumption that it is deployed across symmetric networks. None of these are entirely satisfactory, however, although there are particular scenarios in which each may be worth investigating.

Careful planning of the Condor deployment, with a clear understanding of the underlying network security boundaries may be able to mitigate many of these issues, but there are some scenarios in which firewall/NAT traversal techniques of some description will be required, although the techniques described here all currently have significant security limitations.

More work is required in the area of firewall/NAT traversal techniques, but ultimately many of these issues need to be addressed within the Condor system itself.

Acknowledgements

The authors would like to thank Todd Tannenbaum of the Condor Team for information on Condor's communication patterns. Any errors in describing these in this paper are, of course, our own.

References

[1] What is Condor?: http://www.cs.wisc.edu/condor/description.html

[2] RFC 1918 (Address Allocation for Private Internets): http://www.ietf.org/rfc/rfc1918.txt

[3] Son, Sechang, & Livny, Miron. "Recovering Internet Symmetry in Distributed Computing" (2003). *Proceedings of the 3rd International Symposium on Cluster Computing and the Grid*, Tokyo, Japan, May 2003: http://www.cs.wisc.edu/condor/doc/CCGRID2003.pdf

[4] Condor® Version 6.6.10 Manual, Section 3.1: http://www.cs.wisc.edu/condor/manual/v6.6/3_1Introduction.html

[5] Condor® Version 6.6.10 Manual, Section 3.10.8.2: http://www.cs.wisc.edu/condor/manual/v6.6.10/3_10Setting_Up.html - SECTION004108000000000000000

[6] Kewley, John. Using Condor effectively in the presence of Personal Firewalls (2004): http://tardis.dl.ac.uk/Condor/docs/FW_condor.pdf

[7] Kewley, John. The use of Condor at CCLRC Daresbury Laboratory (2004): http://www.nesc.ac.uk/esi/events/438/CondorActivities/CCLRC_experience.pdf

[8] RFC 1519 (Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy): http://www.ietf.org/rfc/rfc1519.txt

[9] Beckles, B. Building a secure Condor® pool in an open academic environment (2005). *Proceedings of the UK e-Science All Hands Meeting 2005*, Nottingham, UK, 19-22 September 2005, *forthcoming*.

[10] Condor-C: http://www.cs.wisc.edu/condor/manual/v6.7.8/5_4Condor_C.html

[11] GCB (Generic Connection Brokering): http://www.cs.wisc.edu/~sschang/firewall/gcb/index.htm

[12] netfilter/iptables project homepage: http://www.netfilter.org/

[13] DPF (Dynamic Port Forwarding): http://www.cs.wisc.edu/~sschang/firewall/dpf/index.htm