Improving Connection Management of the OpenLDAP Directory Server

Sang Seok Lim, Jong Hyuk Choi, Hubertus Franke
IBM T. J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598, USA
email: {slim,jongchoi,frankeh}@us.ibm.com

Kurt D. Zeilenga
email: kurt@OpenLDAP.org

ABSTRACT
This paper describes our effort to improve the performance of the connection management subsystem of the OpenLDAP directory server. Two proposed architectures, the multi-listener and the lightweight listener architectures will be described and compared to each other in this paper. This paper will also describe our effort to improve the synchronization of multiple threads by introducing semaphore based resource control. This paper shows that our efforts significantly improve the performance of the OpenLDAP directory server in the presence of long latency operations and reduces the overhead caused by unnecessary context switching and inefficient event polling.

KEY WORDS
OS, Internet, LDAP, Connection Management

This paper was submitted for possible publication in the 24th IASTED International Multi-Conference on Applied Informatics (Parallel and Distributed Computing and Networks) 2006.

1 Introduction

LDAP (Lightweight Directory Access Protocol) [4] is one of the key building blocks of Linux information infrastructure together with relational databases. While the data stored in the database is relational information, the data stored in the LDAP server is hierarchical information such as the organizational and the identity data in the small to large scale enterprises and the naming and addressing data in the Internet. In the enterprise IT environment, LDAP is currently being used as the key information repository that stores majority of identity, credential, certificate, policy, and registry information. Figure 1 shows the conceptual architecture of LDAP and RDBMS. Both of them consist of a frontend for request / reply processing and a backend for data storage and retrieval. The LDAP server consists of an LDAP frontend which processes LDAP queries consisting of the base, filter, attribute specification, and other controls and a storage engine backend. The database server consists of an SQL frontend and a storage backend.

The scalability of the LDAP and the database servers are instrumental in establishing scalable information infrastructure in Linux for various enterprise applications and middleware. This paper describes our efforts of improving the connection management subsystem of the OpenLDAP directory server [11] - the infrastructural directory service software in Linux and other open operating systems.

The connection management subsystem in network servers manages multiple concurrent connections to process incoming requests and to respond with the processed results with the following objectives: 1) low latency response, 2) high throughput, and 3) fairness. In order to achieve these key objectives at the same time, it is important to maintain high concurrency among multiple connections. The concurrency can be achieved by implementing multiple execution control flows either by multiple threads [15, 14] or by the event driven programming [1, 2, 13, 5]. When an execution control flow is expected to be blocked waiting for incoming requests or I/O completion, it needs to be scheduled out and other execution control flows are scheduled in to run to achieve high throughput and fairness. Many research efforts have been made to propose and evaluate the most optimized architectures of implementing concurrency.

Like most other network servers, the connection management subsystem sits in the core of the main event processing loop of `slapd`, the OpenLDAP directory server daemon, determining the main programming structure and its scaling behavior. We have started investigating the connection management of OpenLDAP since we found that its current architecture is suboptimal in terms of scalability, resource control, and resilience to long latency operations. We architected two

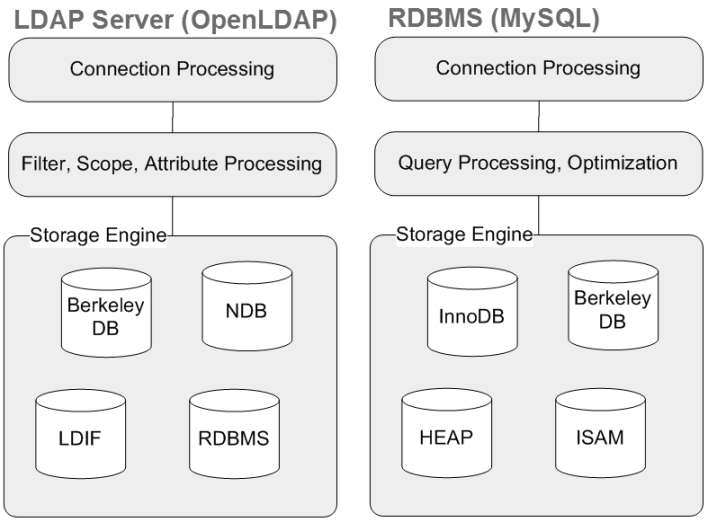


Figure 1: Architecture of Directory and Database Servers.

new designs, the multi-listener and the lightweight listener architectures to improve scalability and latency resilience. We also implemented a semaphore-based resource control scheme to better synchronize multiple threads for reduced context switching overhead and efficient event polling. As a result, the OpenLDAP directory server will be able to support the Linux information infrastructure in a more scalable and robust way.

The rest of this paper is organized as follows. The next section will summarize the existing approaches to achieving high concurrency in the network servers. Section 3 will describe the architecture of the current connection management subsystem of the OpenLDAP directory server and its advantages. Then, we will describe the two new architectures for improved scalability and resilience in detail in Section 4 and their comparative performance evaluation in Section 5. Section 6 will describe the semaphore based resource control scheme and show the reduction in context switches and the resulting improvements in performance. Section 7 will summarize the paper.

2 Highly Concurrent Server Architectures

One of the challenges in building highly scalable LDAP services is dealing with a high level concurrency required to support a large number of clients. There are

three main approaches to achieving high concurrency in network servers including LDAP servers.

2.1 Multi-threaded Server Architecture

The multi-thread architecture relies on the scheduling mechanism of the underlying operating system to achieve concurrency. It further optimizes the multi-process architecture by sharing the address space and hence by reducing the high overhead of context management in the multi-process architecture. The multi-thread architecture provides much better concurrency than the single-threaded architecture since a new thread can continue to process an incoming request should the current thread be blocked waiting for an event. A thread-pool approach optimizes the multi-thread architecture further. A fixed number of worker threads are created in advance and managed as a pool. A listener thread accepts new connections and operation requests and put them into the work queue. The threads in the work thread pool will pick up the work request in the queue as soon as they finish processing the current work.

2.2 Event-Driven Server Architecture

In order to reduce the context switching overhead and the high memory requirement for the stack space in the multi-thread architecture, the event-driven servers rely on the non-blocking I/O primitives and the event dispatching mechanism of the operating systems in order to implement multiple execution control flow in a single thread. Compared to the kernel-thread based multi-thread architecture, the event driven server architecture is more efficient because it does not require context switching. When compared with the user-thread based multi-thread architecture, it does not have the performance problem of the user-level threading upon blocking I/O. One disadvantage of the event-driven architecture is its complicated programming model [15, 14, 9].

2.3 Hybrid Server Architecture

The hybrid server architecture combines the event-driven server to process non-blocking operations with multi-threaded (or multi-process) server to process blocking operations. One of representative hybrid server is the asymmetric multi-process event-driven architecture (AMPED) [12] which combines an event-driven server with multiple helper processes (or threads) that handle blocking disk I/O

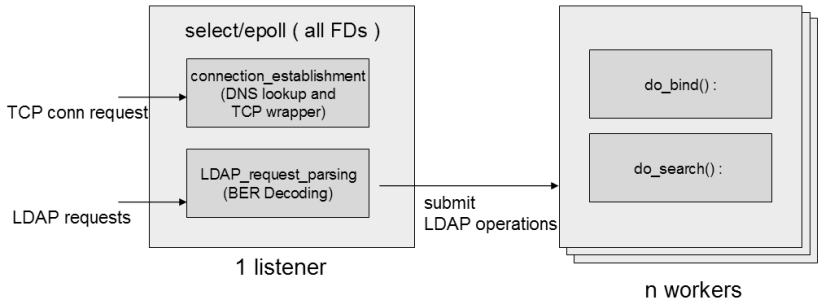


Figure 2: Connection Management Architecture of OpenLDAP **slapd** Directory Server.

operations. Only when a disk operation is necessary, the main event-handling process directs a helper to perform the operation on behalf of the main process. This type of the server performed very well on a variety of workloads and outperformed two aforementioned architectures. But its programming complexity limits its feasibility in real system deployments, making two previous architectures become dominant in the deployment of practical servers.

3 Connection Management in OpenLDAP

The OpenLDAP directory server falls into the multi-threaded server architecture with thread pool. The server consists of a single listener thread and multiple worker threads. The listener takes care of all incoming TCP connection establishment, and parsing and dispatching of incoming LDAP operation requests. The worker threads handle individual LDAP requests by processing attribute filters, accessing the database and writing results back to the clients. The operation of the worker threads can potentially be blocked due to disk or network I/O. The number of maximum worker threads in the pool is fixed. The current connection management architecture of OpenLDAP `slapd` is illustrated in Figure 2. Even though this architecture conforms to the separation of concern between different types of worker tasks and the central listener task and well suited for the requirements from the protocol specification to process multiple operations in a connection simultaneously, it has several disadvantages:

1. The decoding of client requests encoded in BER (Basic Encoding Rules) [6] of ASN.1 (Abstract Syntax Notation One) [7] is quite heavyweight, even

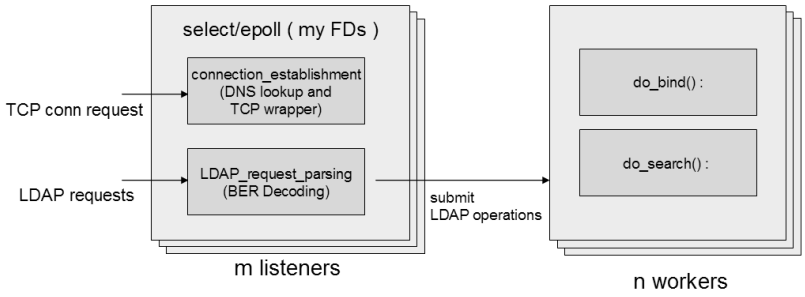


Figure 3: Multiple Listener Thread Architecture.

though LDAP does not use a full fledged BER which of X.500 [8]. This limits concurrency when there are multiple operation requests pending the completion of the BER processing of the current request. These pending latencies will add up to the increased end-to-end latency of LDAP operations.

2. The potentially lengthy operations in connection establishment such as the reverse DNS lookup and the TCP wrapper can significantly degrade the system throughput and operation latency. For instance, if the reverse DNS lookup for a connection establishment take several hundred milliseconds, the server cannot do anything but waiting for the response during that time. When this happens, clients will experience very slow responses.

3. When there are a large number of idle connections present because of the WAN effects and of the clients' request behavior, it has been observed that the server throughput can be degraded because of the overhead of the event dispatching primitives. If this happens in OpenLDAP `slapd`, its effect can be exacerbated by the presence of the heavyweight listener thread.

Because an enterprise scale and the Internet scale LDAP directories are required to store over hundreds of million directory entries and required to support a large number of clients including LDAP proxies, replica LDAP servers, and ordinary LDAP clients, local or remote, it is essential to improve this suboptimal behavior in terms of concurrency of the `slapd`'s connection management subsystem.

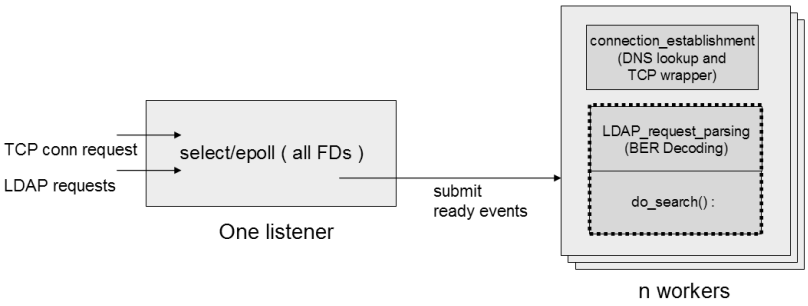


Figure 4: Lightweight Listener Thread Architecture.

4 Two New Architectures

In order to improve the concurrency of the current connection management design of OpenLDAP `slapd`, we architected two new designs. In the first approach, multiple listener threads are processing LDAP requests concurrently so that multiple LDAP requests can be simultaneously processed in spite of the presence of the lengthy operations. On the other hand, the second approach makes the listener threads much lightweight by moving lengthy operations to the actual worker threads. In this section, we will discuss on these architectures in detail.

4.1 Multi-Listener Architecture

As the first approach to improving the concurrency of `slapd`, we architected it to have multiple listener threads. Figure 3 illustrates the architecture of the multiple listener thread design. The single listener thread in the original architecture was cloned to multiple listener threads, each of which is listening to the connections in a partition of the entire connection space. The job of the individual listener thread is identical to that of the original single listener except for the fact that it does not monitor all the connections. It performs the same connection establishment tasks as well as the BER decoding before it hands the operation over to a worker thread. Now that there are multiple of them, however, other listener threads can continue to accept new connections and operations even though some of them are being hit by long latency operations. The number of listener threads is configurable in our implementation.

4.2 Lightweight-Listener Architecture

In the second approach, we architected the single listener thread much lightweight instead of multiplying it. Figure 4 illustrates its architecture. In this design, the listener thread performs the event dispatching only. Tasks such as connection establishment, request reading and decoding are performed in the worker threads before they perform their original tasks, processing of individual operations. Even though there exists only one listener, this architecture does not have the limited concurrency problem as in the original single listener thread architecture, because the lightweight listener thread only blocks on the event dispatching primitives such as `select()` and `epoll()`. The lightweight listener will be able to process the incoming request events even though some of the worker threads are being blocked by long latency operations.

5 Comparative Evaluation of Multi- and Lightweight-Listener Architectures

In this section, we compare the performance of the two new connection management architectures for OpenLDAP in two different scenarios. In the first scenario, we examine the performance characteristics of the two new schemes with the original architecture in the presence of long latency operations in the connection management. In the second scenario, we examine the performance characteristics in the presence of many idle connections to simulate a WAN environment.

The system under test is an IBM xSeries 445 server having 4 2.8GHz Xeon processors and 12GB main memory. The DirectoryMark scripts of Mindcraft were run on an IBM xSeries 445 server having 8 2.8GHz Xeon processors to simulate LDAP clients. The Gigabit Ethernet was used as the connection between the server and the client. The OpenLDAP 2.2 along with Berkeley DB 4.3 was used for the evaluation. The server operating system is SuSE SLES9 with the Linux kernel version 2.6.5.

Figure 5 shows the throughput of the three architectures in the presence of long latency operations. In this experiment, the reverse DNS lookup operations which can take up to 2sec were introduced as the long latency operation. As shown in the figure, the throughput of the original, single threaded listener architecture experiences a serious degradation. It turned out that a slight presence of long latency operations such as the reverse DNS lookup could cause a significant degradation to the performance of the original single threaded listener architecture.

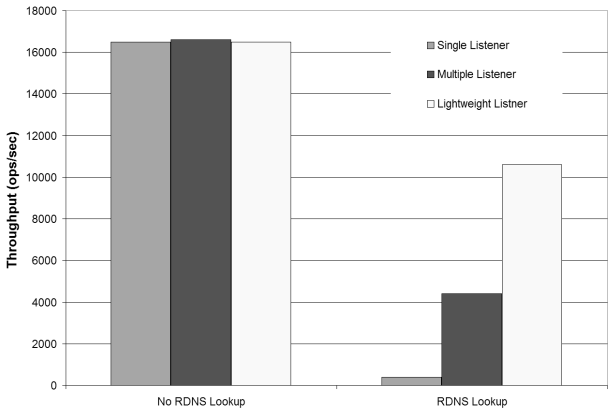


Figure 5: Performance Comparison of New Architectures with Reverse DNS Lookup.

Figure 5 also shows that the throughput of the multi-listener architecture and that of the lightweight listener architecture. The figure shows that the increased concurrency in the multi-listener architecture is able to boost throughput by a significant amount. The throughput improvement of the lightweight listener architecture is turned out to be even more significant. With the lightweight listener architecture, OpenLDAP `slapd` exhibits throughput with the reverse DNS operations as high as over 60% of the original throughput without the reverse DNS operations.

The multi-listener thread architecture does not perform as well as the lightweight listener thread architecture because it requires a complicated file descriptor management scheme and an additional synchronization mechanism to coordinate the execution of the multiple listener threads while the lightweight listener thread architecture does not require such mechanisms.

Figure 6 shows the throughput of the three connection management architectures in the presence of 4000 idle connections simulating the effect of WAN connections which are common in the LDAP servers in the Internet environment. As shown in the figure, both the multi-listener and the lightweight listener thread architectures perform better than the original single listener thread architecture. The increased concurrency in the two new connection management architectures could relieve the overhead of the event dispatching scheme when there exist many idle connections.

Even though the lightweight listener design we implemented has shown a

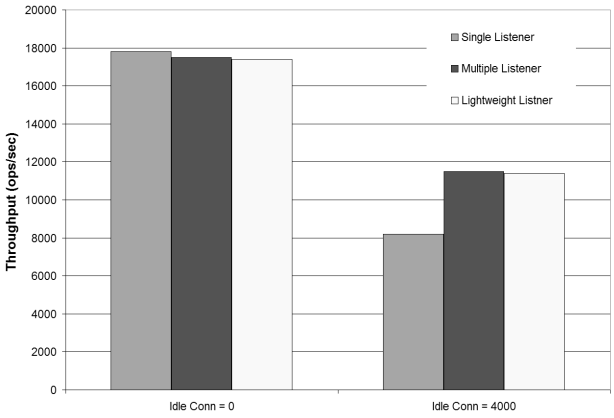
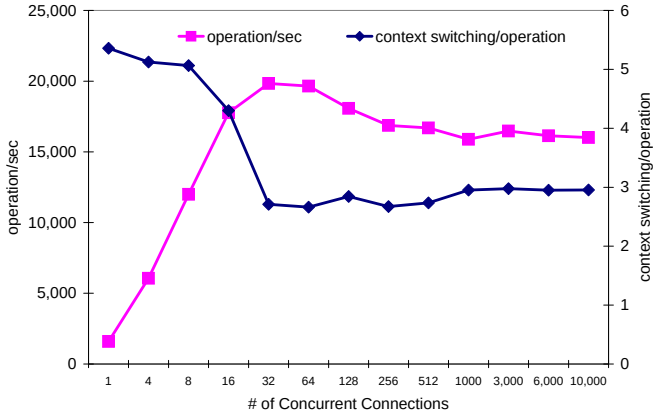


Figure 6: Performance Comparison of New Architectures with Idle Connections.

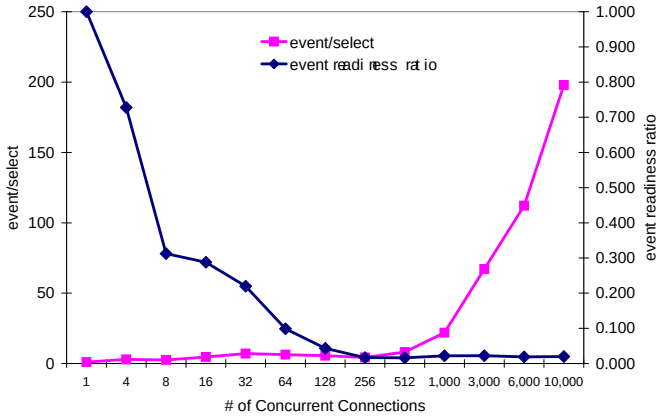
much improved performance in the presence of long latency operations and many idle connections, we observed unnecessary context switches and inefficient event polling during the execution of OpenLDAP `slapd`. This led us to a further optimization in the `slapd`'s threading behavior.

6 Semaphore Based Resource Control

We found that the excessive context switching and inefficient event polling overhead we observed in the lightweight listener thread architecture come from suboptimal coordination of the execution of the listener thread with respect to the available worker threads which is measured and shown in Figure 7. Figure 7 (a) reveals that throughput of the server is degraded as the number of concurrent connections is increased over 32 and context switching occurs three times in average to process a single operation. In Figure 7 (b), event/select and event readiness ratio represent the average number of event-ready connections per each select call and the ratio of event-ready connections over overall connections at a given time, respectively. Even though event/select is increased as the number of concurrent connections is increased, event readiness ratio drops rather steeply. It leads the server to experience high event polling overheads which reaches up to 7.5% of overall CPU time and 20% throughput drop, compared to the peak throughput at 32 concurrent connections. From this measurement, we became convinced that the execution of the listener thread needs to be coordinated in a way to minimize context switching and prevent inefficient event polling.



(a) Context Switching Overhead.



(b) Event Polling Efficiency.

Figure 7: Context Switching Overhead and Event Polling Efficiency.

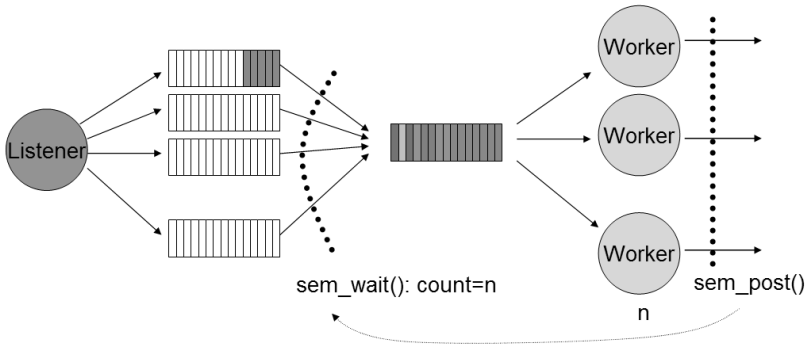
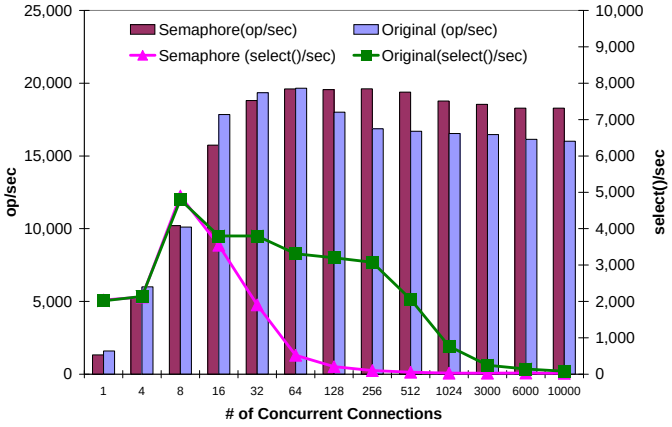


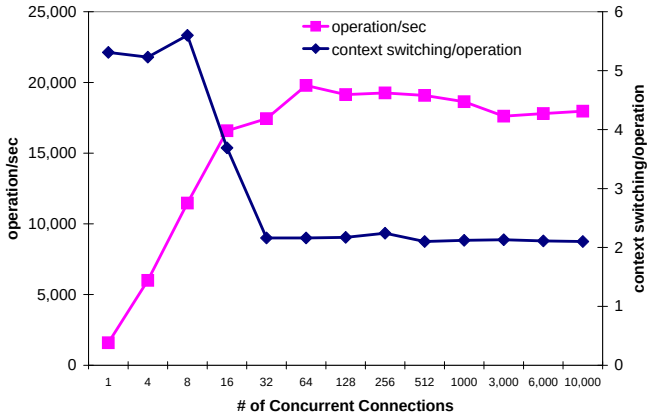
Figure 8: Semaphore Resource Control.

We designed a semaphore-based resource control scheme to better coordinate the execution of the listener thread. Figure 8 illustrates our approach. With the semaphore control, the listener thread waits for a worker thread to become available again by `sem_wait()` with the resource count set to the number of worker threads. When a worker thread becomes available, it notifies the event by `sem_post()`. In this way, it is possible to make the listener thread try to dispatch events only when there are available worker threads to pick up the operation. As a further optimization towards reducing the context switching overhead, multiple worker threads can be treated as a unit of resource allocation. In this scheme, the lightweight listener thread wakes up when as many worker threads as there are in the allocation unit become available at the same time. This further reduces the context switching overhead and prevents frequent event polling.

Figure 9 (a) shows the throughput improvement achieved by the semaphore-based resource control scheme. The bars represents throughput of the original lightweight listener architecture and the one with the semaphore based resource control and the lines represents the rate at which the `select()` primitive is invoked. As shown in the figure, the semaphore based resource control scheme decreases the `select()` rate as the number of concurrent connections increases. As shown in Figure 9 (b) the context switching/operation drops significantly to 2 from 3 without the semaphore control as shown in Figure 7 (a). This directly leads to the reduction in the context switching overhead and improvement in efficiency of event polling by batching more ready events in one select call. As a result, it was possible to obtain more than 16% of throughput gain by implementing the semaphore-based resource control scheme.



(a) Throughput and Event Handling Efficiency.



(b) Context Switching Overhead.

Figure 9: Performance of Semaphore Resource Control.

7 Conclusion

In this paper, we described our efforts of improving the performance of the connection management subsystem of the OpenLDAP directory server by providing much improved concurrency management and semaphore resource control. As a result of our efforts, the OpenLDAP directory server not only becomes much more resilient to long latency operations and many idle connections which are quite common in the Internet environment but also became capable of sustaining much improved throughput. Together with our contributions to the OpenLDAP software in improving the scalability of `slapd` and Berkeley DB [3] and in architecting Component Matching and ASN.1 Awareness [10], our efforts of improving the scalability and resilience of the connection management subsystem of OpenLDAP will significantly improve the enterprise readiness of the OpenLDAP directory services software suite and hence the enterprise readiness of the Linux information infrastructure.

8 Acknowledgements

The authors would like to express special thanks to the OpenLDAP development community for invaluable feedbacks and comments. Special thanks go to Howard Chu of Symas Corp, Pierangelo Masarati of Polytecnico di Milano, and Quanah Gibson-Mount of Stanford University.

References

[1] G. Banga and J. C. Mogul. Scalable kernel performance for Internet servers under realistic loads. In *USENIX 1998 Annual Technical Conference*, 1998.

[2] G. Banga, J. C. Mogul, and P. Druschel. A scalable and explicit event delivery mechanism for UNIX. In *USENIX 1998 Annual Technical Conference*, 1999.

[3] J. H. Choi, H. Franke, and K. D. Zeilenga. Performance of the OpenLDAP directory server with multiple caching. In *Proceedings of International Symposium on Performance Evaluation of Computers and Telecommunication Systems*, July 2003.

[4] J. Hodges, R. Morgan, and M. Wahl. Lightweight directory access protocol (v3): Technical specification. RFC 3377, September 2002.

[5] J. Hu, Pyatali, and D. Schmidt. Measuring the impact of event dispatching and concurrency models on web server performance over high-speed networks. In *the 2nd IEEE Global Internet Conference*, November 1997.

[6] ITU-T Rec. X.690, ASN.1 encoding rules: Specification of basic encoding rules (BER), canonical encoding rules (CER), and distinguished encoding rules (DER), 1994.

[7] ITU-T Rec. X.680, Abstract syntax notation one (ASN.1): Specification of basic notation, December 1997.

[8] ITU-T Rec. X.500, The directory: Overview of concepts, models and service, February 2001.

[9] P. Joubert, R. King, R. Neves, M. Russinovich, and J. Tracey. High-performance memory-based web servers: Kernel and user-space performance). In *USENIX Annual Technical Conference*, 2001.

[10] S. S. Lim, J. H. Choi, and K. D. Zeilenga. Design and implementation of component matching for flexible and secure certificate access in pki. In *Proc. of the 4th Annual PKI R&D Workshop "Multiple Paths to Trust"*, pages 41–51, August 2005. Also published as a NIST Technical Publication, NISTIR 7224, ISBN 1-886843-38-4.

[11] OpenLDAP. `http://www.openldap.org`.

[12] V. S. Pai, P. Druschel, and W. Zwaenepoel. Flash: An efficient and portable web server. In *USENIX Annual Technical Conference*, 1999.

[13] N. Provos and C. Lever. Scalable network IO in Linux. In *USENIX 1998 Annual Technical Conference, FREENIX track*, 2000.

[14] R. von Behren, J. Condit, and E. Brewer. Why events are a bad idea (for high-concurrency servers). In *HotOS Workshop*, May 2003.

[15] R. von Behren, J. Condit, F. Zhou, G. C. Necula, and E. Brewer. Capriccio: Scalable threads for Internet services. In *ACM Symposium on Operating Systems Principles*, 2003.