oTN-2011-01 openlab Summer Student Report

ZODB Benchmarking and Replication

Jurand Nogiec
Supervisor: Jose Benito Gonzalez Lopez
12 August 2011
Version 3
Distribution: **Public**

Abstract

The Indico web-based conference management system needs a mechanism to address replication and performance issues. The aim of this project is to evaluate ZODB replication systems to choose the superior option and enable its use within Indico.

Introduction

Integrated Digital Conference (Indico) is a web-based conference management system and agenda. This software is in use as a production system at CERN and at nearly 100 institutions worldwide. At CERN, it handles on the order of 150,000 total events with 10,000 visitors per day. The back-end for the Indico system is written in Python and uses Zope Object Database (ZODB), a native object database, to store persistent objects. ZODB does not provide a stock method for replication or fail-overs to backup servers and can cause network traffic congestion caused by many simultaneous web requests. The main goal of the work presented in this report is to evaluate two ZODB replication systems: Zope Replication Services (ZRS) and NEOPPOD Distributed Transactional NoSQL Object Database (NEO) versus the current non-replicated ZODB system, select one for use in production, and make its usage within Indico

possible, in order to address the aforementioned issues of replication and traffic bottlenecks. We will start with the configuration supported by the systems, followed by experimental results, and finally documentation of tools and guides for this report.

1 System Properties

1.1 ZRS

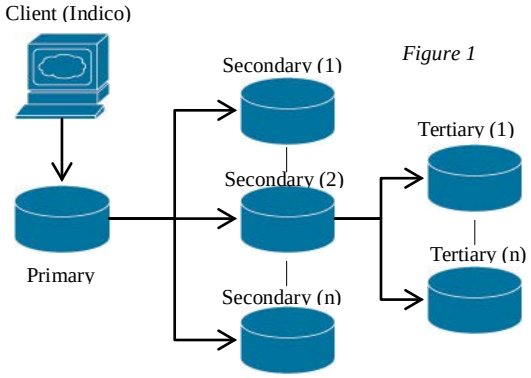


Figure 1

There must be exactly one Primary server to which the client connects (a "Single Master" configuration). The Primary ZRS storage server may replicate to one or more Secondary storage servers. Each Secondary server may replicate to one or more Tertiary servers. In this way the ZRS supports hierarchical configuration. This general setup is illustrated in Figure 1.

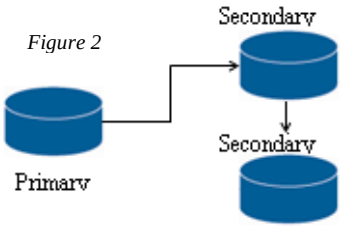


Figure 2

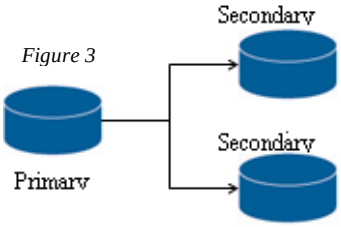


Figure 3

A Secondary (or Tertiary, or Quaternary) server may be in one of two modes: a "forwarding" read-write mode where the server accepts updates from another server and passes it on to another, and a "receiving" mode where the server receives read-only updates.

In Figure 2, we see a "direct" connection setup: the Primary server handles update requests on a listening port and a number of Secondary servers can connect on this port and send requests. In contrast, in Figure 3 we see one Secondary making requests to the Primary server and then relaying any information in an update to the next Secondary node. The former situation is more desirable, as will be seen later in the report, as it has less overhead and performs faster than the latter situation. The other concern is that with the first setup, if one of the Secondary servers fail, the other Secondary is still there to receive replications, which is not the case with the latter setup, as if the Secondary server connected to the Master and the other Secondary fails, the other Secondary must be reconfigured to connect directly to Master server.

1.2 Neoppod

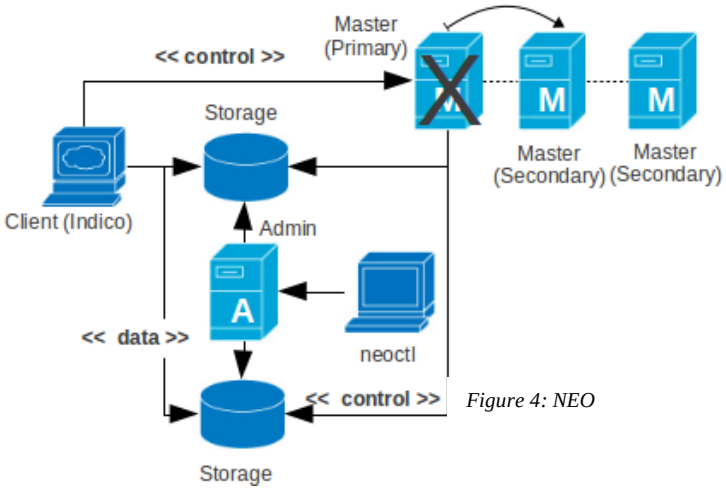


Figure 4: NEO

In Figure 4, a sample configuration of NEO is shown. This configuration has two Storage nodes, secondary and primary Master nodes, and one Administration node. The Administration node dynamically updates and balances load through adding and removing Storage nodes as needed. The secondary Master nodes can

replace the Master node if it is in a fault state, which is known as a multi-master configuration. Multiple Storage nodes are needed so that when one Storage crashes, there is another one to automatically continue storage backup operations. Object write and store operations go directly between the Client (Indico) and Storage nodes while the control data (such as transaction committing) goes directly between the Master Primary node and the Storage nodes.

2 Results

2.1 SSD versus HDD

According to the manufacturer of the SSD, Intel, solid state disks have "extremely high performance […] as compared to standard 10,000 and 15,000 RPM SATA hard drives." [1] This section's aim is to see how accurate this statement is in practical benchmarks.

2.1.1 System Configurations

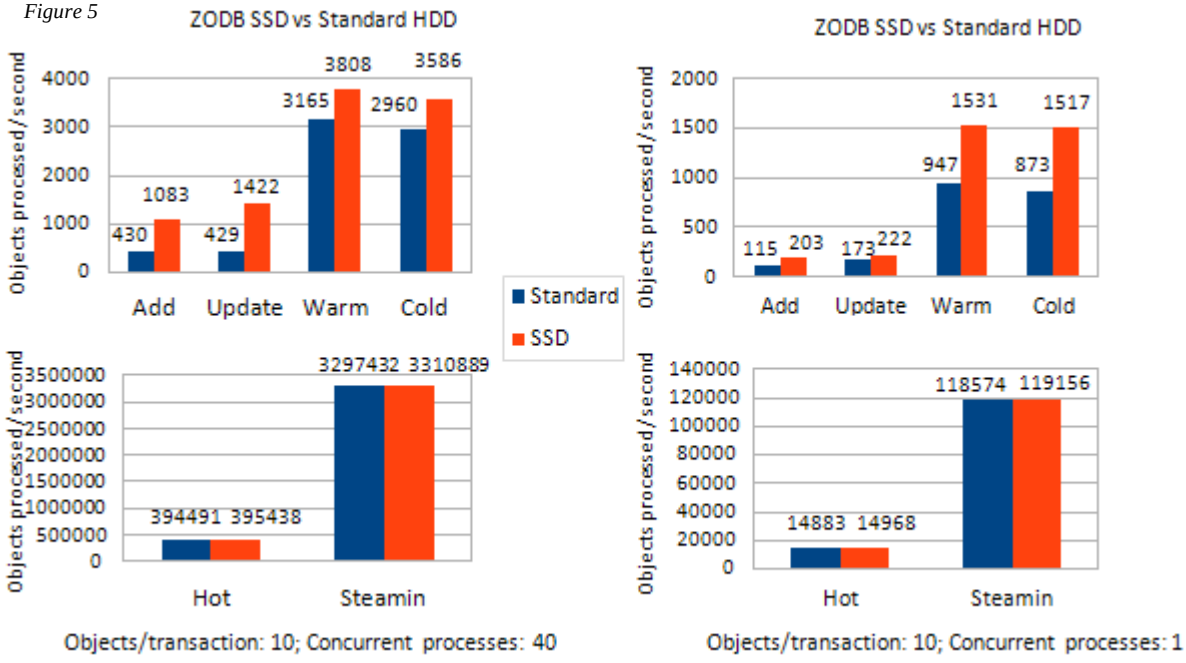
The test-bed for this experiment included two system configurations. One machine housed a solid state disk (SSD) and another had a standard hard disk drive (HDD).

The SSD machine had 12 gigabytes of RAM, an Intel Xeon L5640 processor running at 2.26 gigahertz with a total of 6 cores available. The SSD installed on that machine was an Intel X25-E Extreme SATA Solid-State Drive with storage space of 64 gigabytes. The standard machine had 16 gigabytes of RAM, two Intel Xeon E5410 processors running at 2.33 gigahertz with a total of 8 cores available. The hard disk drive installed on that machine was a Fujitsu Enterprise SCSI interface hard drive running at 10,000 RPM (model number `MBB2147RC`) with storage space of 147 gigabytes. Both systems were running the Scientific Linux CERN SLC (release 5.6) operating system.

2.1.2 Performance

2.1.2.1 Results for Raw ZODB

Figure 5



To understand the above graphs, the following label explanations are necessary:
- *Add* – Start transaction, add a number of persistent objects, then commit transaction.
- *Update* – Change each object added then commit the transaction. Perform no clearing of caches. Transactions are started from same process.

- *Warm* – Read all objects just added. Perform no clearing of caches. Transactions are started from separate processes.
- *Cold (Partial Caching)* – Perform clearing of all caches, then read objects written by Update.
- *Hot* – Clear only the pickle-cache, then read all objects written by Update.
- *Steamin (Full Caching)* – In the same process as Hot, but also use the pickle-cache

These results were obtained using the *zodbshootout* tool [2]. As a summary for these results, use of pickle and other caches minimizes gains from the use of the SSD, as is seen in the Hot and Steamin results, showing nearly the same results for the use of the SSD versus the use of the HDD. The raw data speeds without taking into account caching (the results labeled Add, Warm, and Cold) should be higher, as is the case with synthetic SSD versus Standard HDD testing and it truly tests the performance of the actual drives. The effects of caching do not apply to inputting data into the database (see the results for Update); a cached copy does not help since to write and commit the data, you must still truly write into the database. By changing the amount of concurrent processes to forty, which in previous analysis was determined to be the appropriate amount [3] , we see that the gains from using the SSD drive to house the database file become less extreme, but still significant, in the Add, Update, Warm, and Cold results.

2.1.2.2 Results for Indico-based test

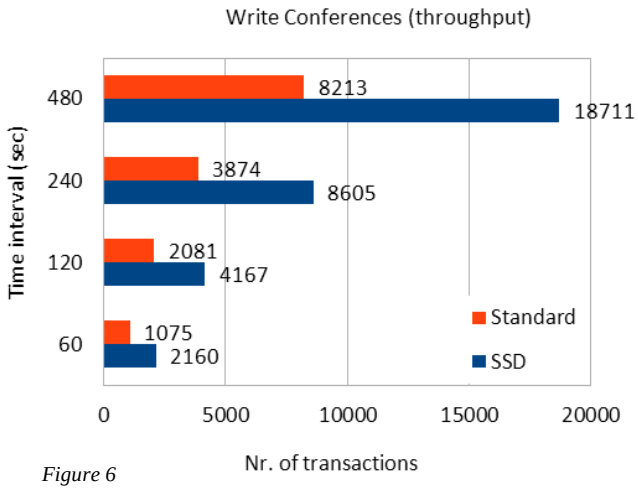


Figure 6

Using the configuration with the SSD has an average of 2.13 times faster performance than using a standard SCSI 10K RPM hard disk drive and a difference of 10498 more transactions within the 480 second result. This trend of double the performance is true at each of the 60, 120, 240, and 480 second throughput results. This is in terms of writing a set of conference objects into the database and measuring the throughput over specified time intervals.

2.1.3 Summary

Using an SSD to host the database files most notably aids in write to database performance and less of an effect in read operations where caching partially cancels the gains of the SSD use.

2.2 ZODB, ZRS and Neoppod

2.2.1 Performance Testing Results

This section is devoted to benchmark testing of the current ZODB system ("Vanilla") versus the ZRS and NEO systems. The methodology and results below are followed by explanations. It is important to note that the Neoppod results in the charts come with a special evaluation ** listed in *2.2.1.4*.

2.2.1.1 Methodology

The results were obtained using the following methodology, using a current 15 gigabyte copy of a flat `Data.fs` Indico database file used in production. Each Category has a list of Conference objects, each Conference has a number of contributions and a meeting room. The read and write tests focused on iterating over these large lists of objects and either reading the value or adding to a value, respectively. A number of trials is used to test the average latency. It measures the time it takes for the object saving in the database to finish. The throughput is calculated by seeing how many read or write transactions can happen per amount of time divided by the amount of time.

In the below graphs, *ZRS (two-direct)* refers to the situation in Figure 3, where the Master server replicates directly to two secondary servers. *ZRS (two-serial)* refers to the configuration in Figure 2, where the Master server replicates transactions to a Secondary server, which then in turn replicates to the next Secondary server. *ZRS (one)* refers to a situation with exactly one secondary node being replicated from the Master. *NEO*** refers to a configuration of NEO with a single storage, administration, and Master nodes. Finally, *Vanilla* refers to the ZODB system with no replication, used as the control variable.

2.2.1.2 Read from database operations

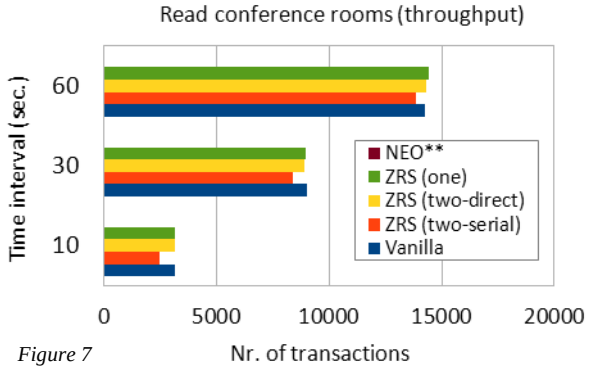


Figure 7

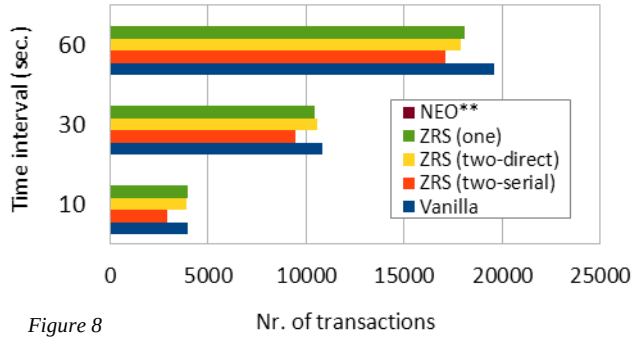


Figure 8

The plain ZODB installation outperformed both the ZRS single and double secondary setups for database read throughput. The throughput for the double secondary scenario was slightly worse than for the single due to the overhead time of replications. The first result is given by reading through all conference records and retrieving the conference room assigned to it. This ensures that the Conference object is truly read under testing. The second result, reading all the

Contributions related to a Conference references a list of other objects. This is used to include the performance of traversing the object tree in the tests.

The throughputs are nearly the same between a single server scenario and double server direct scenario – this makes sense since the master is broadcasting on a port that the secondary servers are communicating with, which should hold no additional overhead.

2.2.1.3 Write to database operations

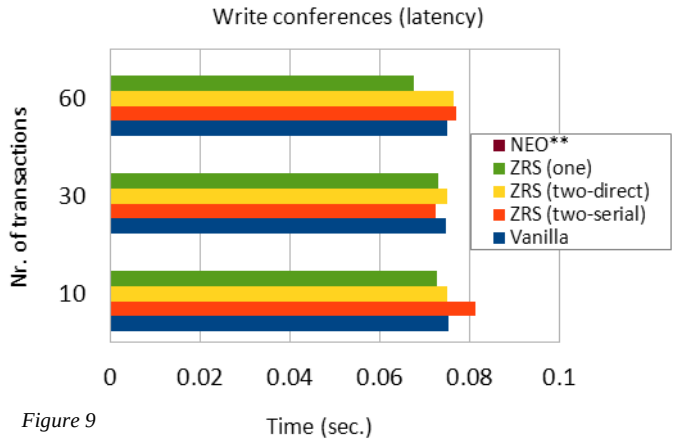


Figure 9

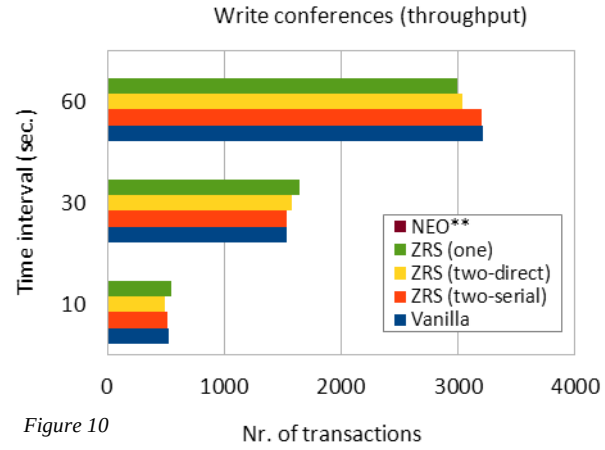


Figure 10

Results of the plain ZODB versus the ZRS solution show that using a double setup slightly increases the latency performance. The range over all values (`R=0.014 sec`) is quite small. The ZRS secondary servers are not synchronous with updates caused by writes. The updates to secondary servers do not happen immediately, but rather they are handled in the style of optimistic (or lazy) replication, which "propagates changes in the background, discovers conflicts after they happen, and reaches agreement on the final contents incrementally." [4] This means the ZRS master does not wait for writes to propagate to the secondary servers before being able to handle the next request, a fact that the results confirm.

There was no significant difference in the throughput performance of the plain and ZRS solutions for write access within the same total time.

2.2.1.4 Neoppod

While using neoppod as the system under test, *there is a significant issue with data consistency errors even under small scale testing* **. During testing while trying to insert Conference and Meeting objects into the database, there were numerous database conflict errors that would leave the system in an unstable and unusable state until a full system reset. The exact error involved with this was called a `Resolvable conflict` in the database. This status message reports that a data corruption error had occurred. These errors were verified with the developers. According to a Neoppod developer from Nexedi, "an object change is missing on a storage,

which is a kind of data corruption" that is reproducible under their own test procedures. He also states that Neoppod is currently being stabilized for possible usage in production systems.

2.2.1.5 Summary

The results of the Nexedi team, along with this report's test results to verify, point to the remaining immaturity of the Neoppod system. It is not yet ready to be used in a production environment. The writes and reads to database results show that the use of the ZRS system generally will provide negligible performance degradation and that the directly connected secondary scenario is the better option versus two serially connected secondary nodes.

2.2.2 Resiliency to Error

2.2.2.1 Steps to Recovery

The ZRS and Neoppod systems have differing steps to recovery, e.g., the master server crashed: what are the steps towards recovering from the system error? Using ZRS, the procedure is to fail over to a secondary backup server manually. A secondary node must be set to be the new primary node in the server configurations, which is a situation similar to recovering a copy of a flat database file by replacing the corrupted database file with a backup, except that this backup will be up-to-date right until the moment of failure. In NEO, the handling of the error is more graceful: another master node takes over for the failing master node automatically when NEO is set-up using a multiple-master node configuration. Without the multi-master configuration, the procedure degrades to that of ZRS's manual fail-over.

2.2.2.2 Expected Downtime

In terms of expected downtime, ZRS and NEO also differ. For ZRS, the downtime is the duration of time in which the Master node is not operational and a secondary node is being selected to do a recovery. There is no inbuilt system to automatically monitor when such storage nodes have failed. Using Neoppod, the downtime is potentially negligible, as the primary master node elects an auxiliary master node. While Storage nodes still need a mechanism to recover after a failure as with ZRS, the Administrator node (through the neoctl utility) gives better feedback as to where failures have occurred.

2.2.2.3 Summary

NEO's multiple master configuration gives it the more streamlined and stable option for recovery, as well as lower expected downtime than the ZRS system.

2.2.3 Other Analysis

2.2.3.1 ZRS

Zope Corporation is a large organization with notable clients including Bank of America and the United States Navy. Theirs is the commercial offering for Zope Replication Services, a ZODB database replication system. In terms of support, the only organization to offer official support for ZRS is Zope Corp itself. ZRS customers are reported to be long term users who have renewed their support licenses numerous times and that there is a number of organizations at one time subjecting the system to evaluation and testing. Their site-wide license costs 40,000 USD and 12,495 USD for the single CPU license. The support is included over the first year, which offers a possibility to evaluate the level of support offered. It costs 25% of total license costs per year thereafter.

2.2.3.2 Neoppod

Neoppod is an open-source system developed by a number of organizations, including Nexedi SA and PilotSystems. Neoppod is the Free Software alternative, licensed under GNU General Public License v2.0. Support is given primarily through answers from the team via mailing list. It is not actively discussed software, as the last listed messages to the mailing list were due to the research directly related to this report. However, representatives from Nexedi were quick to respond to the messages. There does not exist at this time any company offering professional support or consulting for the Neoppod system. Therefore, the costs for support of maintaining

the code and fixing errors in the Neoppod system would be the responsibility of a group that decides to use it, and may realistically cost more in man hours than it does for the ZRS option. This choice may be more in the current open-source spirit of the Indico system, but as it stands ZRS is the better choice for a reliable production system.

2.2.3.3 Summary

Zope's ZRS offering gives a more professionally supported and reliable product than the NEO open-source alternative, but does so at a price premium. However, it is possible that fixing the still immature NEO system and any bugs it introduces may end up costing more Indico team man-hours than ZRS.

2.2.4 Summary

In summary of the results in this section, the use of SSD to store database files primarily improves write performance, NEO is not ready to be used in a production environment and would theoretically provide better recovery options due to its multi-master property, and ZRS provides a more reliable and better supported product and thus should be selected instead of NEO.

3 Documentation

3.1 Tools

The tools listed here can be found in the Indico *git* repository.

3.1.1 Benchmarking

3.1.1.1 zodbtestindico-runner.sh

This is a test runner bash script for Indico ZODB throughput/latency tests . Such a runner is necessary so that each test run separately to ensure the independence of database transactions. The results are dumped to a timestamped directory. It runs the following Python files:

```
zodbtestindico-testReadCategoryThroughput.py
zodbtestindico-testReadConferenceRoomsThroughput.py
zodbtestindico-testReadContributionsThroughput.py
zodbtestindico-testWriteConferencesLatency.py
zodbtestindico-testWriteConferencesThroughput.py
```

3.1.1.2 zodbtest.py

This file is a replication testing base-class for ZODB databases. It performs tests that generate throughput and latency results for raw ZODB transactions not involving the Indico system itself.

3.1.1.3 zodbtestindico.py

This file contains a class for ZODB benchmarks using requests sent to bare Indico instance. It makes available the following testing methods:

`testReadCategoryThroughput`
Tests the amount of category names that can be read over a time interval

`testReadConferenceRoomsThroughput`
Returns the amount of Conference Rooms that can be read in specified amount of seconds.

`testReadContributionsThroughput`
Returns the amount of Contributions per Conference that can be read in specified amount of seconds.

`testWriteConferencesLatency`
Returns the average latency of a Conference modification averaged over amount of repetitions.

`testWriteConferencesThroughput`
Returns the amount of Conferences that can be modified (written to) in specified amount of seconds.

3.1.2 Utilities

3.1.2.1 neo_start.sh

This tool starts a basic configuration (one storage, one administration, one master node). Near the end of the runner script, it keeps re-checking the status of the system (whether it is in recovering or running state) until it changes, which indicates either the ultimate success or failure of the configuration. It is also included in the NEO Installation Guide. It is a bash script that starts the neomaster, neostorage, neoadmin, and finally neoctl programs. They each represent instances of Master, Storage, Administration, and Control nodes, respectively.

3.1.2.2 indicobulkloader.py

This bulk loader is used to generate enough data to simulate an active Indico instance. It inserts a number of Conferences, and for each Conference, adds a number of blank Contributions. To create, for example, a 5 GiB FileStorage Data.fs file, one would need to specify the tool to generate 85,000 empty conferences.

3.1.2.3 zodb.{master, relay, secondary}.conf

These configuration files are used to set up Master, Secondary, and Relay (that both replicate to and from servers) servers, respectively, using the `runzeo` tool.

3.2 Installation Guides

3.2.1 ZRS with Indico

As part of the work done, there now exist two installation guides to show exactly what steps are needed to install ZRS and NEO for use with the Indico system. These guides were used to install a functioning replication environment using a current Indico development server with properties approximate to the production service.
As a summary for the more detailed guide present in AVC's Sharepoint (`zrs_install_guide.html`), the main steps for installation of ZRS for use with Indico are the following:

1. Install ZODB (important to use v3.8.6 due to python2.4 compatibility)
2. Add ZRS python egg to path
3. Modify Indico configuration to point to the Master server
4. Modify ZODB configuration files as in the example files referenced in 3.1.2.3.

3.2.2 Neoppod with Indico

As a summary for the complete guide in Sharepoint (`neo_install_guide.html`), the main steps for installation of Neoppod for use with Indico are the following:

1. Install programs including ZODB
2. Install and configure MySQL as in the standard NEO installation guide
3. Run the custom configuration script
4. Apply patch to Indico code as listed in the complete guide.

3.2.3 Summary

The installation procedures for ZRS and Neoppod are quite similar. The main notes for the procedures are that ZRS has a more straightforward installation than NEO. It also has in its favor the support of zc.buildout automatic configuration tool which is better suited for standardized installation procedures. Neoppod does not support zc.buildout configurations and requires a code change. It also does not have an inbuilt monitoring system as ZRS does. The procedures appear to be functionally rather similar, and one does not have a better procedure than the other.

4 Summary

Considering the use of ZRS versus NEO as the database replication system for Indico, ZRS is the only viable option at this time.

ZRS and NEO have differing configurations and approaches to the replication problem. However, NEO is still too unstable to use in a production system and should not at this time be used for critical applications. The use of ZRS will help to provide a proper way to do replication while minimizing downtime upon system failure versus a non-replicated environment. As an additional result for performance concerns in the system, the use of an SSD for hosting the database files aid primarily write to database performance of the Indico system rather and less of an effect in read operations where caching may cancel the gains of the SSD use. Using the tools and configuration guides, a successful replication environment was established on an Indico development server, and use of the developed test-bed led to the evaluation results given in this report.

5 Bibliography

[1] Galimberti, Davide. "Experimental Performance Analysis in ZODB." Thesis. Ecole Polytechnique Federale De Lausanne, 2010. Print.

[2] Hathaway, Shane. Zodbshootout: A ZODB Performance Test. Python Package Index. N.p., 01 Feb. 2011. Web. 15 July 2011. <http://pypi.python.org/pypi/zodbshootout>.

[3] "Intel® X25-E Extreme SATA Solid-State Drive." Laptop, Desktop, Server and Embedded Processor Technology - Intel. Intel, May 2009. Web. 08 Aug. 2011. <http://www.intel.com/design/flash/nand/extreme/technicaldocuments.htm>.

[4] Saito, Yasushi, and Marc Shapiro. "Optimistic Replication." ACM Computing Surveys 37.1 (2005): 42-81. Print.