ASSESSMENT OF LDAP SERVICES IN HIGH AVAILABILITY ENVIRONMENT

Marcin Marchel[1,2], Cezary Boldak[1]

[1] Faculty of Computer Science, Bialystok University of Technology, Białystok, Poland

[2] Transition Technologies, Warsaw, Poland

Abstract: In this work we assess functioning of an LDAP service in the High Availability Environment. A system configuration with two alternative existing open source LDAP implementations (OpenLDAP and Apache Directory Server) was examined. Pacemaker/Corosync tool were employed here to manage two distributed resources: virtual main IP address and two cloned LDAP services. The test system was deployed on the LDAP production servers at Faculty of Computer Science, Bialystok University of Technology. Then several tests were run to measure the time of various operations (initialization, read/write, switchover, failover) as well as to verify the continuity of working and data consistency in presence of diverse faults, including power supply off. Few low level problems (due to used tools) were encountered and solved.

Keywords: LDAP, high availability, data consistency, Pacemaker/Corosync, OpenLDAP, Apache Directory Server

1. Background

LDAP (Lightweight Directory Access Protocol) [3] is an Internet Protocol that provides access to information that is organized in the hierarchical form (tree-structured). It is also commonly used as an external service for user authentication and authorization in numerous information systems. OpenLDAP [9] and Apache Directory Server [10] are well known open source LDAP implementations. Both of them provide built-in replication functionalities in two modalities: *refresh-only* – where consumers (replicas) periodically synchronize their states with the provider (replicas initiate the synchronization) and *refresh-and-persist* – where the consumers (after initial registration with the provider) are synchronized immediately after the provider is modified (the provider initiates the synchronization).

Advances in Computer Science Research, vol. 12, pp. 21-35, 2015.

High availability (HA) is a term that refers to the information systems with protections against possible unwanted downtime to your system caused by hardware or software malfunctions (faults) [13,14]. The desired availability level (period of correct functioning without failure – statistically estimated or empirically calculated in a longer term) depends on a domain of the system usage, but for the most critical domains on can strive for single seconds of downtime per year (99.99999 % for ultra-avaliability) [5]. "No-dowtime" is probably only the theoretical level.

The key technique for handling such malfunctions is redundancy [4]. According to this assumptions, the defective part of the infrastructure (e.g. server, service) will be automatically replaced with the other spare one available in the pool, which until the fault occurrence might not serve any significant features. When the replaced resource is statefull, all redundant copies must be kept consistent (data replication). But the redundancy is only a pre-requisite to the high availability – further system organization to recognize and deal with faults is necessary to assure better availability rate [5].

High-availability mechanisms refer to functionalities that help to increase its reliability. High-availability mechanisms are:

- **Failover** is the mechanism that performs his job at the time when a lack of access to the resource is detected. This mechanism performs a switch to another available resource that should be mirrored copy of the resource that has failed.
- **Switchover** mechanism is aimed at getting the same as its automatic *failover* equivalent. The difference between these mechanisms lies in the fact that the *switchover* requires the intervention of a person (such as a system administrator) to perform the switch node or restart the software.
- **Switchback** is a mechanism that involves the restoration of network traffic to a node with a higher priority than that which is currently supporting the resource. This mechanism assumes that higher priority node was unavailable for some period of time for any reason.
- **Heartbeat** is the mechanism that for specified period of time (interval) sends information to defined system objects of its availability. Depending on the information returned by this mechanism defined actions will be performed.

There exist technologies/tools to facilitate implementation of the high availability paradigm. One of them is Pacemaker/Corosync [11,12], offering low level solutions (heartbeat, distributed resource definition and management) to the the high available system developer .

2. Related works

The literature study did not reveal many works on assuring the high availability in hierarchical databases (LDAP). [1] proposes a solution (BFT-LDAP) to overcome Byzantine Faults to keep all the LDAP replicas consistent, but does not take into account the necessity of the high availability of the service, for instance when a physical machine is gone. [2] presents a very detailed study on the OpenLDAP performance (time measures for several LDAP operations) in different hardware and software configurations in order to achieve the optimum response time and throughput. But only one OpenLDAP instance was examined, not replicated nor distributed, and without component faults, what is of the main concern in this work.

Much more sources treat high availability in standard relational databases. [6] describes a highly available configuration of the MySQL database using Heartbeat, rsync and internal MySQL replication functionality, but no verification of the proposed solution was performed. [7] builds a "highly available" PostgreSQL configuration in the cloud using the Threshold Based File Replication technique. The server replication and automatic, load-balanced request routing seem to fulfill the HA requirements, but more interests there is oriented toward the load-balancing concerns (practical experiments) while the system resiliency in presence of faults is not studied at all. [8] also proposes a highly available transactional database system (using Oracle and Tuxedo tools). This work focuses on redundant components (hardware), database replication, parallel server/clustering and transactional replication. Several test scenarios (even complex ones) were prepared and performed. They consisted of: database failure, server process crash, network failure and verified the correct functioning of the system in their presence. The response times for 10,000 messages were measured (however not reported in details), but not times of other operations (e.g. failover).

3. Test system configuration

Our test system is based on redundancy of the run LDAP servers, controlled by the the Pacemaker/Corosync software suite, to provide the high availability of the LDAP service and fault tolerance (verified experimentally in Section 4.) to several malfunctions. The redundant LDAP servers are kept synchronized by the internal mechanisms of the two analyzed LDAP implementations.

The system needs at least two machines (real or virtual), but can be further grown up to a larger number, to take into account more failed nodes. In our experimental configuration we used two nodes:

– **ldap-one** - Main node with *IP1*,
– **ldap-two** - Backup node with *IP2*.

The two IP addresses *IP1* and *IP2* are not exposed to clients – they are used only for internal resource management. Instead, the third address *IPglobal*, bound to a DNS name, is offered to external systems (by means of the bound DNA name) – see below.

The Pacemaker/Corosync environment (installed in the distributed manner on all nodes) defined two managed resources.

1. **Virtual IP Address** (*IPglobal*): This resource was configured with higher priority assigned to the ldap-one node. It means that in case of 2 nodes availability, the *IPglobal* address will be assigned to **ldap-one** node (which is the real machine, while **ldap-two** is the virtual one). This resource was working in Active/Passive mode.
2. **LDAP Resource**: This resource (representing the run and redundant LDAP servers) had to be cloned for every node. It means that it should be active all the time on every node defined in the cluster (working in Active/Active mode). There was some additional work done by us to create a new OCF (Open Cluster Framework) [15] resource agent for Apache Directory Server, because it was not provided out of the box. OCF resource agent for OpenLDAP is available as out of the box solution for *Pacemaker*. Creating new resource agent for *Pacemaker* made us possible to perform examinations for both ApacheDS and OpenLDAP implementations.

Detecting a node failure (loss) and switching to the backup node are managed with the Pacemaker/Corosync suite but recovering to the initial (fully operational) configuration can be more complicated. Some faults (local LDAP process, system restart) are to be dealt with the used tool (by restarting, what is sufficient in many cases [5]), but others (system stop, power off, machine error) need a manual intervention. In this case, after detecting the permanent loss of one node, the system should report this abnormal state to the administrator. Such behavior can be enabled by adding *ocf:pacemaker:ClusterMon* resource. This resource provides notifying about cluster events, which can be received in a few different ways like email/SNMP (Simple Network Management Protocol) notifications or by using external agent (shell script).

Two different LDAP implementations were examined independently: OpenLDAP and Apache Directory Server. Both of them were configured accordingly to their technical documentation to the *refresh-and-persist* replication. This mode is closer to assure the strong data consistency (all modifications of the main node are immediately propagated to the backup ones), while the second mode (*refresh-only*)

realizes the eventual data consistency (changes are propagated periodically, so with a latency) [4].

Figure 1 presents functioning of the test system. External users (systems) use the domain name resolved by the DNS to *IPglobal*. This address, as the managed distributed resource, is assigned to the only one node from the pool. It is worth noticing that the entire system works in the Active/Pasive model and the redundant replicas do not serve the client requests. It could be further changed to the Active/Active model, implementing some load balancing technique [14] to increment the system throughput. The connection between the LDAP resources means the replication mechanism.

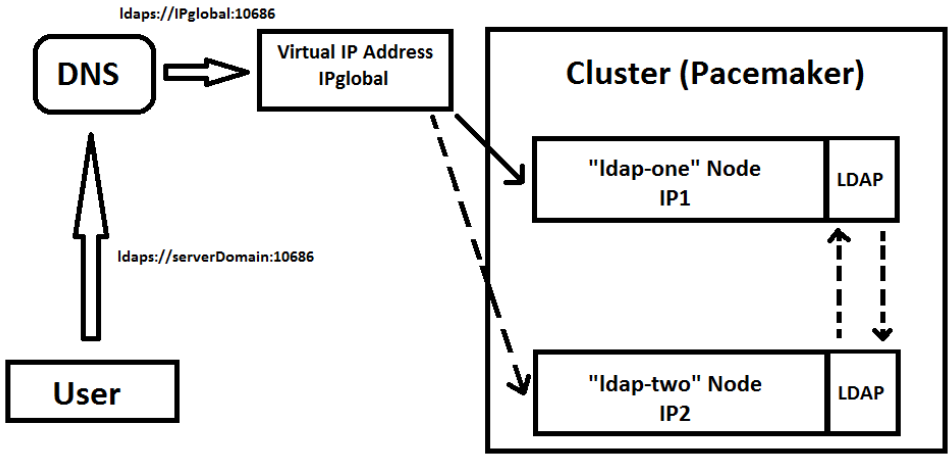

Fig. 1. Architecture of the test system.

4. Experiments

The main goals of the prepared tests were to:

- verify the failover functioning under different malfunctions,
- measure the response time of basic LDAP operations, in normal conditions and in presence of faults,
- verify the data consistency after the failover server switch.

All the tests were performed on the two selected LDAP implementations (OpenLDAP 2.4 and Apache Directory Server 2.0.0-M20) independently, so the ad-

ditional goal was to compare these two technologies regarding all the three mentioned above aspects.

The test environment was set up on the production servers at the Faculty of Computer Science, Bialystok University of Technology, Poland. A cluster of two nodes was built, one located on a real machine, second on a virtual one. A third machine in the same network was used to run all the test scripts. Tests were performed without using the LDAPS secure protocol. All experiments were run 10 times (except stated differently) and min, max and mean statistics are presented.

Test plan

We planned and ran four types of tests.

1. **Measuring times of read and write operations in normal circumstances.** The main goal was to have the reference for the following experiments (scaling). Additionally the read and written information was located on different tree levels to check if it influenced the response times.
2. **Measuring times of initialization/restoration of the cluster functioning** after a single node loss and in case of the entire cluster restarting.
3. **Verifying the failover functioning and measuring its times**. Only one node (active) failure was simulated. As this mechanism is driven by the heartbeat messages, experiments were conducted for several their intervals.
4. **Verifying the data consistency in the event of active node failure**. LDAP unavailability was caused by different reasons: killing the LDAP service local process, controlled node disconnection from the cluster, controlled machine shutdown, pulling out the power plug.

4.1 Read/write times (no faults present)

Tests presented in this chapter are designed to check how fast it is possible to save and read a specified data set from/to the LDAP server. This examination also assumes that no node failure would not occur during performing it. An important aspect of this examination is the depth of the tree where the information is stored: 10, 30 50, 100.

Read times

This examination assumed preforming 10,000 read operations. Every read operation was done on a different entry from the set of 1,000 entries and on the specified tree depth to minimize usage of the memory cache mechanism. The previously mentioned

entries were located at four different depths of the tree structure (10, 30, 50, 100). Every such the depth contained its own set of 1,000 entries. We used default cache settings for both ApacheDS and OpenLDAP. The algorithm shown below presents steps of our shell script to calculate time required to perform 10,000 read operations:

1. **Start** the time counter.
2. In the **loop** of 10,000 iterations:
   (a) generate DN for the current loop iteration,
   (b) execute read operation (*ldapread*) – repeat it until success if failed occasionally.
3. **Stop** the time counter. Calculate time required for performed operations and display result.

Results in Table 1 present times for both LDAP implementations and take into account different depths of the tree structure. Based on the examination it can be concluded that OpenLDAP offers 7% higher read speed than ApacheDS. It is also worth noticing that depth of information in a tree structure does not significantly affect the speed of reading.

Table 1. Read times (in seconds) in function of the tree depth (no faults).

Depth of a tree structure	10		30		50		100	
	ADS	OpenLDAP	ADS	OpenLDAP	ADS	OpenLDAP	ADS	OpenLDAP
Minimum	203.49	188.81	204.99	192.15	209.39	192.33	222.16	197.08
Maximum	211.21	195.14	211.07	202.46	219.42	197.20	230.37	206.87
Average	206.08	193.03	207.55	195.36	212.89	194.73	225.36	200.49

Write times

This examination assumed performing 10,000 write operations. As before, the writes modified data at different depth of the tree structure. This script assumes existence of the defined entry on the specified tree depth, which contains *Description* attribute. Every write operation performs incrementation of that attribute's value. The algorithm given below presents steps of our shell script to calculate the time required to perform 10,000 write operations:

1. **Start** the time counter.

2. In the **loop** of 10,000 iterations:
   (a) generate DN for the write operation using the current loop iteration context,
   (b) perform the write operation (*ldapmodify*) – repeat it until success if failed occasionally.
3. **Stop** the time counter. Calculate time required for performed operations and display result.

Results in Table 2 present times for both LDAP implementations and take into account different depths of the tree structure. Based on the examination it can be concluded that OpenLDAP implementation is able to write information about 38 times faster than ApacheDS. It can be partially acceptable (in the case of ApacheDS) in scenarios where the data is much more frequently read then written. Here again, the tree depth has only minor influence on the performance.

Table 2. Write times (in seconds) in function of the tree depth (no faults).

Depth of a tree structure	10		30		50		100	
	ADS	OpenLDAP	ADS	OpenLDAP	ADS	OpenLDAP	ADS	OpenLDAP
Minimum	7410.86	193.12	7317.77	209.62	7423.12	205.68	7653.84	209.97
Maximum	7498.54	240.16	7351.10	241.95	7478.83	239.48	7714.01	237.50
Average	7455.31	213.42	7331.34	224.85	7450.69	224.08	7685.88	227.71

4.2 Times of initialization/restoration the cluster functionality

The examinations presented in this chapter were executed to verify how fast will the developed system restore its full availability after occurrence of two different fault types: loss of all the nodes and loss of the active node. The algorithm shown below presents steps that were done for both test cases. The script implementing it is executed on the external node.

1. In a **loop** wait until LDAP becomes inaccessible - it is performed by executing *ldapsearch* command and checking the returned status. In meanwhile the manual step is done – pacemaker/corosync services are restarted on the active node.
2. **Start** the time counter.
3. *ldapsearch* command is executed in a second **loop** finished once LDAP becomes accessible again.
4. **Stop** the time counter and print a message with the calculated time period.

After a failure of all the nodes

That examination concerned the system behavior after failure of all two available nodes (forced cluster software issue). The algorithm presented above was executed in the normal cluster configuration, where two nodes had active (by cloning) LDAP resource. Data in Table 3 presents the examination results for two chosen LDAP implementations. Based on them, it can be concluded that OpenLDAP becomes accesible 40% faster than ApacheDS. The latter showed a high variance, with observations going from 22.11 to 52.49.

Table 3. Times (in seconds) needed to restore LDAP functionality (all nodes failed in the same time).

LDAP	ADS	OpenLDAP
Minimum	22.11	23.88
Maximum	52.49	25.38
Average	39.27	24.38

After a failure of the active node

That examination concerned the system behavior after failure of the only one available node that hosted the LDAP resource. The algorithm presented above was executed in the abnormal cluster configuration, where only one node was working (e.g. after a *failover* or *switchover* event). Data in Table 4 presents examination results for two chosen LDAP implementations. Based on the examination it can be concluded that OpenLDAP becomes accesible about 40% faster than ApacheDS.

Table 4. Times (in seconds) needed to restore LDAP functionality (active node failed).

LDAP	ADS	OpenLDAP
Minimum	40.39	23.69
Maximum	42.80	32.48
Average	41.03	25.51

4.3 Examination of the failover functioning

This examination verified one of the main issues of this work: if the automatic switching to the backup node (*failover*) functioned at all. It also involved setting various time intervals for the heartbeat module to examine its influence of this mechanism speed. This test assumed that LDAP process would be manually killed on the active node. Measured time showed how fast LDAP resource became available again. Used algorithm shown below presents steps that were done (the script implementing it was executed on the external computer).

1. In a **loop** wait until LDAP becomes inaccessible, its performed by executing *ldapsearch* command and checking returned status – in meanwhile the manual step is done: LDAP process (or 2 processes in case of ApacheDS) is killed on the active node .
2. **Start** the time counter.
3. *ldapsearch* command is executed in a second **loop** finished once LDAP becomes accessible again, what is verified by use of *ldapsearch* command.
4. **Stop** the time counter an print a message with the calculated time period.

Data placed in Table 5 presents examination results for the chosen LDAP implementations. Based on the examination it can be concluded that:

- *failover* worked all the times,
- heartbeat period affected the *failover* time in nearly linear manner (Figure 2),
- LDAP implementation did not affect the *failover* time.

Table 5. Times (in seconds) needed to restore LDAP functionality in function of the heartbeat period.

Heartbeat configuration	3		10		15		20	
	ADS	OpenLDAP	ADS	OpenLDAP	ADS	OpenLDAP	ADS	OpenLDAP
Minimum	0.59	0.18	2.31	0.25	1.28	0.29	2.03	4.94
Maximum	2.98	3.04	9.12	9.88	13.29	14.89	23.19	28.66
Average	1.49	1.93	5.19	5.58	6.86	7.29	9.47	15.77

4.4 Examination of the data consistency after a failure of the active node

After confirming the *failover* functioning, this examination aimed probably the most important goal for the proposed solution: verification of the data consistency when

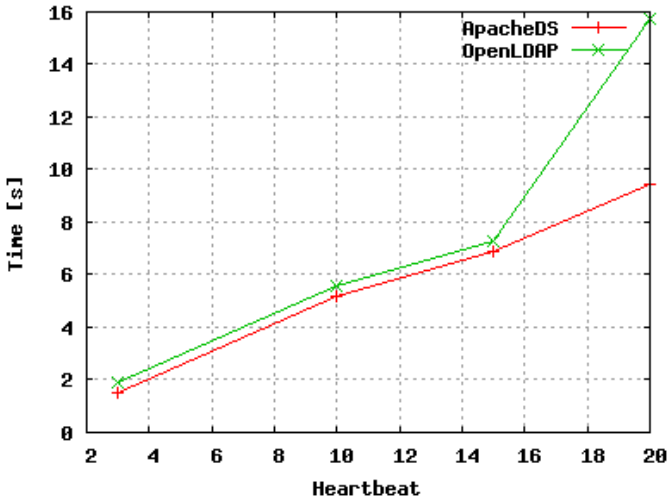

Fig. 2. Failover times in function of the heartbeat period.

a fault (consisted of a failure of the active node) occurred during the data sending process (*refresh-and-persist* replication model was used).

This test was performed in four variants differing in the malfunction nature:

- **Controlled disconnecting of the main node from the cluster**, which was performed by restarting the Pacemaker/Corosync services on the active node.
- **Killing the LDAP process**, which was performed by executing a shell script that was able to automatically find LDAP process in memory and kill that process (two processes in case of ApacheDS) on the active node.
- **Controlled server shutdown**, which was perfomed using *shutdown* command on the active node.
- **Plugging off the power plug**, which was performed by manually plugging off the power plug on the active node.

We used the algorithm presented below and composed of the following steps.

1. **Set** a local counter (*LC*) variable value to 1.
2. **Write** *LC* (*ldapmodify*) to the LDAP tree - it becomes a remote counter (*RC*).
3. In a **loop** (in meanwile one of the faults described above is produced):
   (a) **read** *RC* and **calculate** delta $D = LC - RC$ (it shows how many data modifications were not replicated due to the active node failure – if it occurred in that loop iteration),

(b) **if** $D \neq 0$ **then**: **display** a message about the event and **exit** the shell script,
(c) **increment** *LC*,
(d) **increment** *RC* (by *ldpapmodify*).

The results in Table 6 presents examination results for two chosen LDAP implementations. There was no data loss/inconsistency detected – our system for the both chosen LDAP implementations **passed this test**.

Table 6. Numbers of the data loss cases (data inconsistency).

Fault	Controlled disconnecting node from cluster		Killing LDAP process		Controlled server shutdown		Pulling the power plug	
	ADS	OpenLDAP	ADS	OpenLDAP	ADS	OpenLDAP	ADS	OpenLDAP
Minimum	0	0	0	0	0	0	0	0
Maximum	0	0	0	0	0	0	0	0
Average	0	0	0	0	0	0	0	0

4.5 Encountered problems

We believe that an important aspect of our work is also to inform about problems that were encountered in the studies.

We noticed an unexpected replication mechanism behavior during performing examinations for both selected LDAP implementations. This problem occurred in case of disconnecting power supply for the server and then turning the server on again. The problem was caused by the replication mechanism being broken – it stopped working bidirectionally. The only way to fix this issue was to restart LDAP on the node, which was previously working as the backup one. This problem occurred when the *refresh-and-persist* replication mode was chosen. In case of the *refresh-only*, this problem was not occurring. The workaround for it was to extend initially created OCF resource script for ApacheDS to be able to detect such situation and perform restarting cluster services on the node that caused the problem.

Another type of issue that was noticed during the testing was a problem with commands like *ldapsearch* and *ldapmodify*. This issue was about hanging process in case when destination LDAP server got offline during command execution. In such the case an additional parameter related to the process timeout was not working as it was expected. This issue was noticed only for the ApacheDS implementation. The workaround for it was to use eternal shell script that was used to detect and kill hanging *ldapsearch* or *ldapmodify* process.

5. Conclusions

This work resulted in the experimental assessment of functioning of the cluster system realizing the highly available LDAP service, in normal working and in presence of different faults . These examinations, thanks to using two different LDAP implementations, allowed us to make more general observations about behavior of the LDAP services in High Availability Environment in similar configurations.

Performed operations aimed to experimentally demonstrate the most important features of high availability on example of the developed system. They responded to the question whether the selected LDAP implementations are able to maintain consistency of data between server instances in the case of a node failure. In the case of any of the examined implementations data loss has not occurred at the time of the main node failure. In our opinion, in the case of large production environments where read/write operations from/to LDAP resource can be counted in thousands per second, the result may be different than the one obtained in our studies. Particularly in the case of a decidedly slower write operation when using the ApacheDS implementation.

On the basis of the studies we additionally described four characteristics, which in our opinion defines perfect LDAP implementation oriented for high availability.

1. **The quality of replication mechanism**: this is an important feature which can ensure that data consistency will be kept in case of cluster's node loss.
2. **The speed of writing information**: this is an important feature for the replication mechanism because it can directly cause to not replicate the information in the case of node failure (provider), which updates the information to other nodes (consumers).
3. **The speed of reading information**: this is an important feature for replication mechanism when you need to query the LDAP instance for specific information.
4. **The speed of the service startup**: this is particularly important in the case of configuration, when a node becomes available only at the time of the root node loss. In such case, the time it takes to boot up the LDAP server is the time which delays the start of the entire cluster environment.

On the basis of those studies, we observed that the OpenLDAP implementation is more stable in the case when high availability is required. However, during our experiments, negative influence of the lower ApacheDS performance have not been encountered, but we do not exclude such a possibility in a longer time period usage.

This work can be continued in several directions. More LDAP implementations can be examined to prove the generality of the system. The system availability can

be even increased by including (and further experimental verification) more backup nodes. The system performance can be boosted by routing client requests to all the nodes, not only the main one, with the load balancing techniques [14]. It would be also interesting to observe and measure the system behavior and availability in longer term (months or years) to empirically classify our solution to the concrete availability class [5].

Acknowledgment

This work was supported by the Bialystok University of Technology with the grant S/WI/2/2013.

References

[1] Ali Shoker, Jean-Paul Bahsoun: Towards Byzantine Resilient Directories, IEEE 11th International Symposium on Network Computing and Applications, 2012.

[2] Xin Wang, Member, Henning Schulzrinne, Fellow, Dilip Kandlur, and Dinesh Verma: Measurement and Analysis of LDAP Performance, IEEE/ACM Transactions On Networking, Vol. 16, No. 1, pp. 232–243, 2008.

[3] Franco Milicchio, Wolfgang Alexander Gehrke: Distributed Services with OpenAFS. Springer Science & Business Media, 2007.

[4] Andrew S. Tanenbaum, Maarten Van Steen: Distributed Systems. Principles and Paradigms (2nd Edition). Prentice-Hall, 2006.

[5] J. Gray, D.P. Siewiorek: High-availability computer systems. Computer, Vol. 24, No. 9, pp. 39–48, 1991.

[6] V. Chaurasiya, P. Dhyani, S. Munot: Linux Highly Available (HA) Fault-Tolerant Servers. Information Technology, (ICIT 2007). 10th International Conference on, pp. 223–226, 2007.

[7] S. Pippal, S. Singh, R.K. Sachan, D.S. Kushwaha: High availability of databases for cloud. Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on pp. 1716–1722, 2015.

[8] S. Budrean, Y. Li, B. C. Desai: High Availability Solutions for Transactional Database Systems. Proceedings of the Seventh International Database Engineering and Applications Symposium (IDEAS'03) 2003.

[9] OpenLDAP Software 2.4 Administrator's Guide, Site: `http://www.openldap.org/doc/admin24/OpenLDAP-Admin-Guide.pdf`, Access time: 19 October 2015

[10] ApacheDS 2.0 Advanced User Guide, Site: `https://directory.apache.org/apacheds/advanced-user-guide.html`, Access time: 19 October 2015

[11] Andrew Beekhof: Pacemaker 1.1, Configuration Explained, An A-Z guide to Pacemaker's Configuration Options, Site: http://clusterlabs.org/doc/en-US/Pacemaker/1.1-pcs/pdf/Pacemaker_Explained/Pacemaker-1.1-Pacemaker_Explained-en-US.pdf, Access time: 19 October 2015

[12] Andrew Beekhof: Pacemaker 1.1, Clusters from Scratch, Step-by-Step Instructions for BuildingYour First High-Availability Cluster, Site: http://clusterlabs.org/doc/en-US/Pacemaker/1.1-pcs/pdf/Clusters_from_Scratch/Pacemaker-1.1-Clusters_from_Scratch-en-US.pdf, Access time: 19 October 2015

[13] Theo Schlossnagle: Scalable Internet Architectures Sams Publishing, 2006.

[14] P. Membrey, E. Plugge, D. Hows: Practical Load Balancing, Ride the Performance Tiger Apress, 2012.

[15] Florian Haas: The OCF Resource Agent Developer's Guide, Site: http://www.linux-ha.org/doc/dev-guides/ra-dev-guide.html, Access time: 20 October 2015

OCENA USŁUG KATOLOGOWYCH LDAP W ŚRODOWISKU WYSOKIEJ DOSTĘPNOŚCI

Streszczenie W niniejszej pracy oceniono działanie usług katalogowych LDAP w środowisku wysokiej dostępności. Wzięto pod uwagę dwie implementacje LDAP o otwartym kodzie: OpenLDAP i Apache Directory Server. W celu zarz adzania dwoma rozproszonymi zasobami (wirtualny adres IP i dwie sklonowane usługi LDAP) użyto narzędzia Pacemaker/Corosync. Testowa konfiguracja został wdrożona na serwerach produkcyjnych LDAP na Wydziale Informatyki Politechniki Białostockiej. W dalszej kolejności przeprowadzono testy w celu pomiaru czasów różnych operacji (inicjalizacji, odczytu/zapisu, zaplanowanego i awaryjnego przeł aczenia serwerów) jak również w celu zweryfikowania ci agłości pracy i spójności danych w przypadku różnego rodzaju awarii, w tym zaniku zasilania. Konieczne było rozwi azanie kilku technicznych problemów zwi azanych z użytymi narzędziami.

Słowa kluczowe: usługi katalogowe LDAP, wysoka dostępność, spójność danych, Pacemaker/Corosync, OpenLDAP, Apache Directory Server

Artykuł zrealizowano w ramach pracy badawczej nr S/WI/2/2013.