



**EDB™ Postgres High Availability &
Horizontal Read Scaling Architecture**

Version 1.0

March 21, 2019

EDB Postgres High Availability & Horizontal Read Scaling Architecture, Version 1.0
by EnterpriseDB® Corporation
Copyright © 2019 EnterpriseDB Corporation. All rights reserved.

EnterpriseDB Corporation, 34 Crosby Drive, Suite 201, Bedford, MA 01730, USA
T +1 781 357 3390 **F** +1 978 467 1307 **E** info@enterprisedb.com **www**.enterprisedb.com

Table of Contents

1 Introduction	1
1.1 EFM Overview.....	2
1.2 PgPool-II Overview.....	3
1.2.1 PCP Overview.....	4
1.2.2 Pgpool Watchdog.....	4
2 Architecture	5
3 Components Required to Implement High Availability with PgPool	8
3.1 EFM Configuration File	8
3.2 Pgpool Configuration File	9
3.3 pgpool_backend.sh	10
4 Optional Components	11
4.1 Virtual IP Addresses	11
4.2 Pgpool Watchdog.....	11
Appendix A: Supported Failover Scenarios	13
Appendix B: Integration Scripts	14
B-1 load_balancer_detach.sh.....	14
B-2 load_balancer_attach.sh.....	14
B-3 follow_master.sh.....	14
B-4 pgpool_backend.sh	15

1 Introduction

Since high-availability and read scalability are not part of the core feature set of EDB Postgres Advanced Server (also referred to as EDB Postgres, or simply EPAS), EPAS relies on external tools to provide this functionality, namely EDB Failover Manager (EFM) and Pgpool-II (commonly referred to simply as Pgpool). The document provides a proposed architecture showing the components that are used in order to provide this functionality and discusses the implications of architectures formed around these tools. This document demonstrates how to integrate EFM and Pgpool to leverage the benefits they both provide. Using the reference architecture described in Section 2, users can learn how to achieve high availability with automatic failover mechanism (using EFM) while scaling the system for larger workloads and a high number of concurrent clients with read-intensive or mixed workloads to achieve horizontal scaling/read-scalability (using Pgpool).

The architecture described in this document has been developed and tested for EFM 3.3, pgPool II 4.0 and Advanced Server 11.

Documentation for Advanced Server and Failover Manager are available from EnterpriseDB at:

<https://www.enterprisedb.com/de/docs/en/3.2/edbfm/toc.html>

Documentation for pgPool-II can be found at:

<http://www.pgpool.net/docs/latest/en/html>

1.1 EFM Overview

EDB Postgres Failover Manager (EFM) is a high-availability module produced by EnterpriseDB to monitor the health of Postgres Streaming Replication clusters and verify failures quickly. When a database failure occurs, EFM can automatically promote a streaming replication standby node into a writeable master node to ensure continued performance and protect against data loss with minimal service interruption.

Basic EFM Architecture Terminology

- An EFM cluster is comprised of EFM processes that reside on the following hosts on a network:
- A **master node** is the primary database server that is servicing database clients.
- One or more **standby nodes** are streaming replication servers associated with the master node.
- A **witness node** confirms assertions of either the master or a standby in a failover scenario. A cluster does not need a dedicated witness node if the cluster contains three or more other nodes; if a third cluster member that is a database host is not available, a dedicated witness node can be added to the cluster.

1.2 PgPool-II Overview

Pgpool-II (Pgpool) is an open source application that provides connection pooling and load balancing for horizontal scalability of SELECT queries on multiple standbys in EPAS and community Postgres clusters. Pgpool can be configured to use a `backend_weight` parameter to prevent read traffic to be directed to the master node. In such cases, data modification language (DML) queries (i.e., INSERT, UPDATE, and DELETE) are always sent to the master node, while read queries are load-balanced to the standbys, providing scalability with mixed and read-intensive workloads.

More comprehensively, Pgpool provides the following features:

Connection Pooling

Pgpool saves connections to the Postgres and EPAS servers, and reuses them whenever a new connection with the same properties (i.e. username, database, protocol version) comes in. It reduces connection overhead, and improves system's overall throughput.

Replication

Pgpool can manage multiple Postgres and EPAS servers. Using the replication function enables creating a realtime backup on 2 or more physical disks, so that the service can continue without stopping servers in case of a disk failure.

Load Balancing

If a database is replicated, executing a SELECT query on any server will return the same result. Pgpool takes advantage of the replication feature to reduce the load on each Postgres and EPAS server by distributing SELECT queries among multiple servers, improving system's overall throughput. At best, performance improves proportionally to the number of Postgres and EPAS servers. Load balance works best in a situation where there are a lot of users executing many queries at the same time.

Limiting Exceeding Connections

There is a limit on the maximum number of concurrent connections with Postgres and EPAS, and connections are rejected after this many connections. Setting the maximum number of connections, however, increases resource consumption and affects system performance. Pgpool also has a limit on the maximum number of connections, but extra connections will be queued instead of returning an error immediately.

1.2.1 PCP Overview

Pgpool provides a interface called PCP for administrators to perform management operation, such as getting Pgpool status or terminating Pgpool processes remotely. PCP commands are UNIX commands which manipulate Pgpool via the network.

1.2.2 Pgpool Watchdog

`watchdog` is an optional subprocess of Pgpool to provide a high availability feature to itself. Features added by `watchdog` include:

- Health checking of pgpool service
- Mutual monitoring of other watchdog processes
- Changing active/standby state in case of certain faults detected
- Automatic virtual IP address assigning synchronous to server switching
- Automatic registration of a server as standby in recovery

More information about the Pgpool `watchdog` component can be found at:

<http://www.pgpool.net/docs/latest/en/html/tutorial-watchdog.html>

2 Architecture

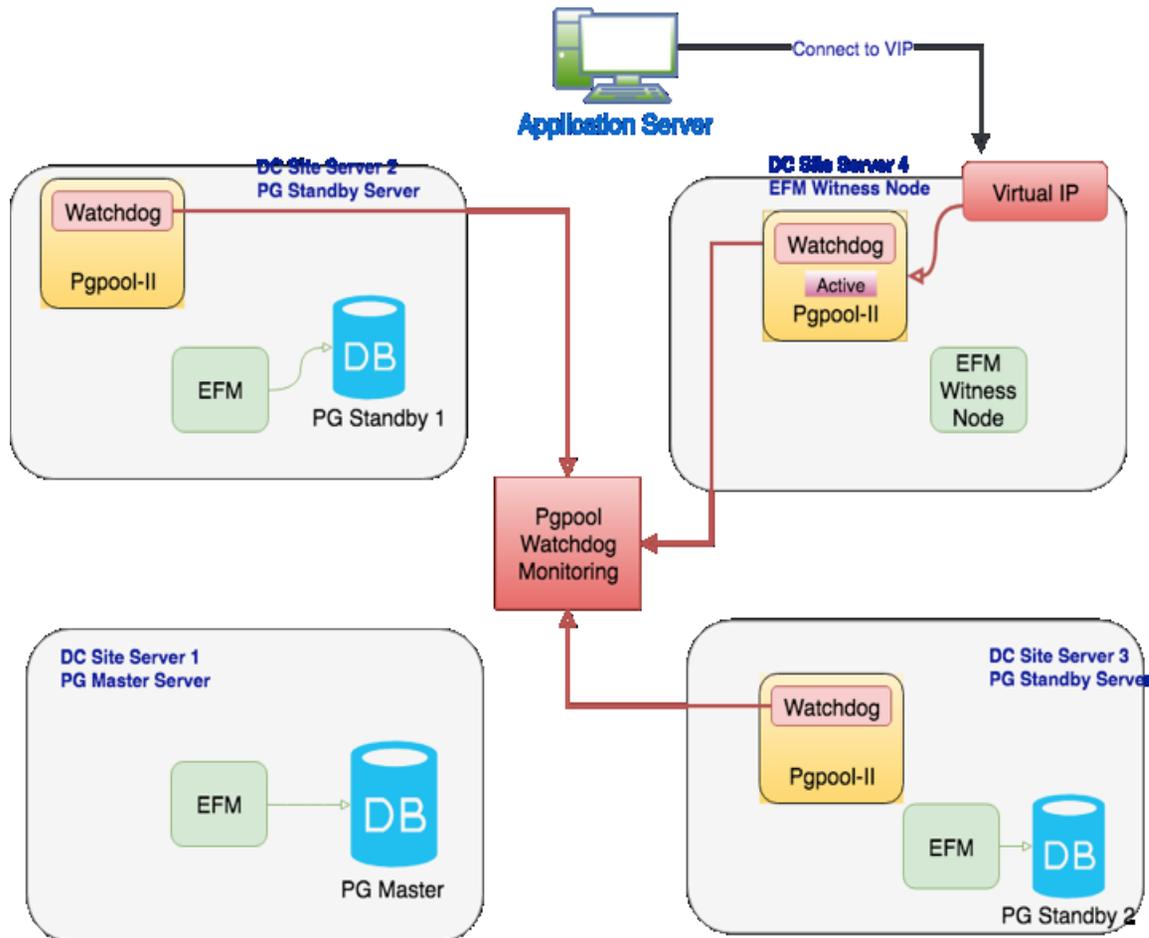


Figure 2.1 - A typical EFM and PgPool configuration

The sample architecture diagram (Figure 2.1) shows four nodes as described in the table below:

Scenario	Components
Server1	EDB Postgres Master, EFM 3.2 This is the EPAS Streaming Replication master node. Applications/clients will connect to this database via the Pgpool port on Server 4 (or via a Virtual IP, if used) to perform write operations (i.e., INSERT, UPDATE, DELETE).
Server2 & Server3	EDB Postgres Standby 1 & 2, EFM 3.2, Pgpool-II (optional) This is an EPAS Streaming Replication standby node. Applications/clients will connect to this database via the Pgpool port on Server 4 (or via a Virtual IP, if used) to perform read operations (i.e., SELECT). An optional standby Pgpool instance can be set up here with watchdog running, if desired.
Server4	Pgpool-II, EFM 3.2 (optional, in witness mode) This server is set up with no active database, but with EFM 3.2 in witness mode, and with Pgpool. All applications/clients will connect to other databases via this server, either via this server's IP address, or via a Virtual IP that points to this server. Note, as mentioned in Section 1.1.1, a witness node is not required if at least three other

	EFM nodes exist. The witness node in this sample architecture is provided for demonstration purposes.
--	-------------------------------------------------------------------------------------------------------

This architecture:

- achieves maximum availability by providing two standbys in case of master node failure.
- achieves maximum performance with mixed and read-intensive workloads by introducing increased read scalability with more than one standby for load balancing.
- reduces load on the master node by performing load balancing and not running Pgpool on the master.
- avoids single point of failure of Pgpool by configuring Pgpool in HA mode using watchdog.
- runs Pgpool master/active instance on the least-burdened node (the EFM witness node) to boost performance while sharing resources with EFM on Server4 (to reduce TCO).
- If one or more standbys are configured with synchronous replication, users can achieve near-zero data loss in a failure event.

With this architecture, you can expect the following behavior:

Scenario	Impact on HA	Impact on Read Scalability
<p>Switchover/Switchback This is a planned downtime taken for some OS/DB level activities and when promoting any of the available standby as master during downtime.</p>	<p>No impact on HA (except couple of seconds of disturbance during role change). Number of nodes in the EFM/PgPool cluster are intact. Switchover will be done by EFM (via an EFM command). One of the available standbys will be promoted as the new master during the downtime. The old master will be reconfigured as a new standby by EFM w/o manual intervention to maintain the total number of nodes in HA setup.</p>	<p>No impact on read scalability(except couple of seconds of disturbance during role change). After switchover the total number of standbys in the cluster will remain same so no impact on load balancing/read -scalability. Once the switchover is done by EFM, it will call post promotion script to update the PgPool with changes; accordingly PgPool will change the role of all the cluster nodes.</p>
<p>Failover This is unplanned downtime which can occur, making the master database inaccessible to an Application (Primary DB down).</p>	<p>No Impact on HA, although such an incident (Failover) leaves only one standby in the EFM/PgPool cluster. To maintain the maximum availability (1 master, 2 standbys) at all times, the old/downed master must be rebuilt as a new standby (either through pg_basebackup or pg_rewind) and attached to the EFM cluster (or a new machine should be introduced as a standby). Requires manual intervention by DBA.</p>	<p>Read scalability will be impacted. Only one standby will be available for read-scalability/load balancing after the failover, until the old/downed master is rebuilt as a new standby and attached to the EFM cluster. Once are total number of nodes (three nodes in this case) are restored, the EFM attach script will attach the node with Pgpool</p>

	Failover will be performed automatically by EFM.	cluster. After completion, both standbys are available for load balancing.
--	--------------------------------------------------	----------------------------------------------------------------------------

3 Components Required to Implement High Availability with PgPool

The health of Postgres nodes is monitored by EFM, and in the event of a master node failure EFM will perform automatic failover to a standby node. Note that Pgpool does not monitor the health of backend nodes and will not perform failover to any standby nodes.

Beginning with version 3.2, EFM agents can be configured to use Pgpool's PCP interface to detach the failed node from Pgpool load balancing after performing failover of the standby node. More details about the necessary configuration file changes and relevant scripts will be discussed in this section.

3.1 EFM Configuration File

EFM 3.2 provides integration with Pgpool to remove failed Database nodes from Pgpool load balancing. EFM 3.2 can also re-attach nodes to Pgpool when returned to the EFM cluster. This is achieved through the configuration of load balancer attach and detach scripts in the `efm.properties` file, using the following parameters:

- `script.load.balancer.attach=/path/to/load_balancer_attach.sh %h`
- `script.load.balancer.detach=/path/to/load_balancer_detach.sh %h`

The script referenced by `load.balancer.detach` is called when EFM decides that a database node has failed. The script then detaches the node from Pgpool by issuing a PCP interface call. Users may verify a successful execution of the `load.balancer.detach` script by calling `show nodes` in a `psql` session attached to the Pgpool port, which will reveal that the node is marked as `down`. This verifies that Pgpool will not send any queries to the downed node.

The script referenced by `load.balancer.attach` is called when a previously-downed node comes back online and is added back to the EFM cluster. Note that a `resume` command must be issued to the `efm` command-line interface to add the node back to the EFM cluster. Once this is done, the script referenced by `load.balancer.attach` is invoked, issuing a PCP interface call, which adds the node back to the Pgpool cluster. Users may verify a successful execution of the `load.balancer.attach` script by calling `show nodes` in a `psql` session attached to the Pgpool port, which will reveal that the node is marked as `up`. This verifies that Pgpool will resume using this node as a load balancing candidate. Sample scripts for each of these parameters are provided in Appendix B.

3.2 Pgpool Configuration File

On the Pgpool side, the following parameters need to be configured in `pgpool.conf`:

```
follow_master_command = '/path/to/follow_master.sh %d %P'

load_balance_mode = on
master_slave_mode = on
master_slave_sub_mode = 'stream'

fail_over_on_backend_error = off
health_check_period = 0
failover_if_affected_tuples_mismatch = off
failover_command = ''
failback_command = ''

search_primary_node_timeout = 3
```

When the primary/master node is changed in Pgpool (either by failover or by manual promotion) in a non-EFM setup, Pgpool detaches all standby nodes from itself, and executes the `follow_master_command` for each standby node, making them follow the new master node. Since EFM reconfigures the standby nodes *before* executing the post-promotion script (where a standby is “promoted” to primary in Pgpool to match the EFM configuration), the `follow_master_command` merely needs to reattach standby nodes to Pgpool.

Note that the load-balancing is turned on to ensure read scalability by distributing read traffic across the standby nodes

Note also that the health checking and error-triggered backend failover have been turned off, as EFM will be responsible for performing health checks and triggering failover. It is not advisable for pgpool to perform health checking in this case, so as not to create a conflict with EFM, or prematurely perform failover.

Finally, `search_primary_node_timeout` has been set to a low value to ensure prompt recovery of Pgpool services upon an EFM-triggered failover.

3.3 pgpool_backend.sh

In order for the attach and detach scripts to be successfully called, a `pgpool_backend.sh` script must be provided. `pgpool_backend.sh` is a helper script for issuing the actual PCP interface commands on Pgpool-II. Nodes in EFM are identified by IP addresses, while PCP commands are sent to Pgpool via a node ID. `pgpool_backend.sh` provides a layer of abstraction to perform the IP address to node ID mapping transparently.

4 Optional Components

4.1 Virtual IP Addresses

Both Pgpool-II and EFM provide functionality to employ a virtual IP for seamless failover. While both provide this capability, it must be noted that EFM associates virtual IP to the master database node while Pgpool associates a virtual IP to the currently -active Pgpool host (in a multi-Pgpool watchdog setup).

Note that if an active instance of Pgpool (Server 4 in our sample architecture) goes down, any available standby Pgpool instance (according to watchdog priority) will take charge as the active Pgpool instance.

4.2 Pgpool Watchdog

Watchdog provides high availability for Pgpool nodes. This section lists the configuration parameters required to configure watchdog on each Pgpool node.

Common Watchdog Configuration Parameters for All Pgpool Nodes

```
use_watchdog = on          # enable watchdog
wd_port = 9000             # watchdog port, can be changed
delegate_IP = 'Virtual IP address'
wd_lifecheck_method = 'heartbeat'
wd_interval = 10 # we can lower this value for quick detection
wd_life_point = 3

# virtual IP control
if_cmdconfig_path = '/sbin' # ifconfig command path
if_up_cmd = 'ifconfig eth0:0 inet $_IP_$ netmask 255.255.255.0'
                                     # startup delegate IP command
if_down_cmd = 'ifconfig eth0:0 down' # shutdown DIP
arping_path = '/usr/sbin'          # arping command path
arping_cmd = 'arping -U $_IP_$ -w 1' # arping command
```

Custom Watchdog Configuration Parameters for Each Pgpool Node

The following configuration parameters must be set on each Pgpool node. The interval and retry values can be adjusted depending upon the requirements and testing results.

```
other_pgpool_hostname0 = '<server# IP/hostname>'
other_pgpool_port0 = 9999
```

```
other_wd_port0 = 9000
other_pgpool_hostname1 = '<server# IP/hostname>'
other_pgpool_port1 = 9999
other_wd_port1 = 9000
wd_priority = <any integer>
```

Note that `wd_priority` can be used to elevate the local watchdog node priority in the elections to select master watchdog node. The node with the higher `wd_priority` value will get selected as master watchdog node when cluster will be electing its new master node at the time of cluster startup or in the event of old master watchdog node failure.

Appendix A: Supported Failover Scenarios

A summary of supported failover scenarios is provided below. Please note that these summaries are not comprehensive, and users should consult the EFM documentation for detailed information about how EFM handles each failover/failure scenario:

https://www.enterprisedb.com/docs/en/3.2/edbfm/EDB_Failover_Manager_Guide.1.29.html

Scenario	EFM/pgPool Response
Master Database is Down	In most cases, EFM will perform a failover by promoting one of the available standbys into a master node. Virtual IP addresses (if configured) will be re-assigned.
Master agent crashes or master node fails	To prevent premature failover/promotion, EFM will first check to see if the master database is still in service (i.e., only the EFM agent on the master server is down, not the entire machine). If necessary, EFM will subsequently perform a failover by promoting one of the available standbys into a master node. Virtual IP addresses (if configured) will be re-assigned. For more information, see: https://www.enterprisedb.com/docs/en/3.2/edbfm/EDB_Failover_Manager_Guide.1.29.html
Standby agent exits or standby node fails	EFM will notify the administrator and invoke the <code>load_balancer_detach.sh</code> script (when properly configured in <code>efm.properties</code>). For more details on <code>load_balancer_detach.sh</code> script please see <code>load_balancer_detach.sh</code> in Appendix B
Dedicated witness agent exits or node fails	EFM: Administrator is notified Pgpool: Pgpool will perform Pgpool failover and one of the existing standby PgPool instances will be promoted as the active instance. Virtual IP (if configured) will be re-assigned to new active instance
Standby gets added back to EFM cluster	When a standby node comes back up, it gets added back to the EFM cluster by using the <code>efm resume <clustername></code> command. The <code>load_balancer_attach.sh</code> script is subsequently called (when properly configured), and updates the Pgpool cluster via the PCP interface. For details on this, please see <code>load_balancer_attach.sh</code> in Appendix B

Appendix B: Integration Scripts

B-1 load_balancer_detach.sh

```
#!/bin/bash
#%h host name
output_file=/tmp/efm-scripts/pp_log
pool_backend=/tmp/efm-scripts/pgpool/pgpool_backend.sh
node_address=$1
current_date_time=`date +"%Y-%m-%d %H:%M:%S"`;

echo $current_date_time >>$output_file
echo "node address to detach = $node_address". >>$output_file
$pool_backend detach $node_address >>$output_file

echo "-----".>>$output_file
exit 0
```

B-2 load_balancer_attach.sh

```
#!/bin/bash
#%h host name
output_file=/tmp/efm-scripts/pp_log
pool_backend=/tmp/efm-scripts/pgpool/pgpool_backend.sh
node_address=$1
current_date_time=`date +"%Y-%m-%d %H:%M:%S"`;

echo $current_date_time >>$output_file
echo "node address to attach = $node_address". >>$output_file
$pool_backend attach $1 >>$output_file
echo "-----".>>$output_file
exit 0
```

B-3 follow_master.sh

```
#!/bin/sh
PCP_USER=                # PCP user name
PCP_PORT=                # PCP port number as in pgpool.conf
PCP_HOST=                # hostname of Pgpool-II
PGPOOL_PATH=            # Pgpool-II installed path
export PCPPASSFILE=     # Path to PCPPASS file

# Execute command by failover.
# special values:  %d = node id
```

```

#           %h = host name
#           %p = port number
#           %D = database cluster path
#           %m = new master node id
#           %M = old master node id
#           %H = new master node host name
#           %P = old primary node id
#           %R = new master database cluster path
#           %r = new master port number
#           %% = '%' character
detached_node_id=$1
old_master_id=$2

echo detached_node_id $1
echo old_master_id $2

## If $detached_node_id is equal to $old_master_id,
## then do nothing, since the old master is no longer
## supposed to be part of the cluster.

if [ $detached_node_id -ne $old_master_id ]; then
    sleep 10
    $PGPOOL_PATH/pcp_attach_node -w -U $PCP_USER -h $PCP_HOST -p
$PCP_PORT $detached_node_id
fi

```

B-4 pgpool_backend.sh

```

#!/bin/bash
#
# pgpool-II backend node configuration driver.
#
# usage: promote_standby.sh hostname [port]
#
# set the following variables according to your setup

PCP_USER=           # PCP user name
PCP_PORT=           # PCP port number as in pgpool.conf
PCP_HOST=           # hostname of Pgpool-II
PGPOOL_PATH=       # Pgpool-II installed path
export PCPPASSFILE= # Path to PCPPASS file

# function returns the Pgpool-II backend node-id for the given
hostname

```

```

# and port number, And if the node-id is not found 255 is
returned
# Arguments:
# 1- Hostname
# 2- Port (optional) if not provided, node-id of first matching
#           hostname will be returned
#
function get_pgpool_nodeid_from_host {
    if [ -z "$1" ]; then
        echo "hostname not provided"
        return 255
    fi

    #Now get the total number of nodes configured in Pgpool-II
    node_count=`$PGPOOL_PATH/pcp_node_count -U $PCP_USER -h
$PCP_HOST -p $PCP_PORT -w`
    echo searching node-id for $1:$2 from $node_count configured
backends
    i=0
    while [ $i -lt $node_count ];
    do
        nodeinfo=`$PGPOOL_PATH/pcp_node_info -U $PCP_USER -h
$PCP_HOST -p $PCP_PORT -w $i`
        hostname=`echo $nodeinfo | awk -v N=1 '{print $N}'`
        port=`echo $nodeinfo | awk -v N=2 '{print $N}'`
        #if port number is <= 0 we just need to compare hostname
        if [ "$hostname" == $1 ] && ( [ -z "$2" ] || [ $port -eq
$2 ] ); then
            echo "$1:$2 has backend node-id = $i in Pgpool-II"
            return $i
        fi
        let i=i+1
    done
    return 255
}

# function returns 1 if Pgpool-II backend node for the given
hostname
# and port number is the primary node in Pgpool-II
# returns 0 for the standby node and 255 if no node exist for the
hostname
# Arguments:
# 1- Hostname
# 2- Port (optional) if not provided, node-id of first matching
#           hostname will be returned

```

```

#
function is_host_is_primary_pgpool_node {
    if [ -z "$1" ]; then
        echo "hostname not provided"
        return 255
    fi

    #Now get the total number of nodes configured in Pgpool-II
    node_count=`$PGPOOL_PATH/pcp_node_count -U $PCP_USER -h
$PCP_HOST -p $PCP_PORT -w`
    echo searching node-id for $1:$2 from $node_count configured
backends
    i=0
    while [ $i -lt $node_count ];
    do
        nodeinfo=`$PGPOOL_PATH/pcp_node_info -U $PCP_USER -h
$PCP_HOST -p $PCP_PORT -w $i`
        hostname=`echo $nodeinfo | awk -v N=1 '{print $N}'`
        port=`echo $nodeinfo | awk -v N=2 '{print $N}'`
        role=`echo $nodeinfo | awk -v N=6 '{print $N}'`
        #if port number is <= 0 we just need to compare hostname
        if [ "$hostname" == $1 ] && ( [ -z "$2" ] || [ $port -eq
$2 ] ); then
            echo "$1:$2 has backend node-id = $i in Pgpool-II"
            # check if the node role is primary
            if [ "$role" == "primary" ]; then
                return 1
            else
                return 0
            fi
        fi
        let i=i+1
    done
    return 255
}

# Function promotes the node-id to the new master node
# Arguments:
# 1- node-id: Pgpool-II backend node-id of node to be promoted
to master
function promote_node_id_to_master {
    if [ -z "$1" ]; then
        echo "node-id not provided"
        return 255
    fi
}

```

```

    $PGPOOL_PATH/pcp_promote_node -w -U $PCP_USER -h $PCP_HOST -p
$PCP_PORT $1
    return $?
}

# Function attach the node-id to the Pgpool-II
# Arguments
# 1- node-id: Pgpool-II backend node-id to be attached
function attach_node_id {
    if [ -z "$1" ]; then
        echo "node-id not provided"
        return 255
    fi
    $PGPOOL_PATH/pcp_attach_node -w -U $PCP_USER -h $PCP_HOST -p
$PCP_PORT $1
    return $?
}

# Function detach the node-id from the Pgpool-II
# Arguments
# 1- node-id: Pgpool-II backend node-id to be detached
function detach_node_id {
    if [ -z "$1" ]; then
        echo "node-id not provided"
        return 255
    fi
    $PGPOOL_PATH/pcp_detach_node -w -U $PCP_USER -h $PCP_HOST -p
$PCP_PORT $1
    return $?
}

# function promotes the standby node identified by hostname:port
# to the master node in Pgpool-II
# Arguments:
# 1- Hostname
# 2- Port (optional) if not provided, node-id of first matching
#     hostname will be promoted
#
function promote_standby_to_master {
    get_pgpool_nodeid_from_host $1 $2
    node_id=$?
    if [ $node_id -eq 255 ]; then
        echo unable to find Pgpool-II backend node id for $1:$2
        return 255
    else

```

```

        echo promoting node-id: $node_id to master
        promote_node_id_to_master $node_id
        return $?
    fi
}

# function attaches the backend node identified by hostname:port
# to Pgpool-II
# Arguments:
# 1- Hostname
# 2- Port (optional) if not provided, node-id of first matching
#           hostname will be promoted
#
function attach_node {
    get_pgpool_nodeid_from_host $1 $2
    node_id=$?
    if [ $node_id -eq 255 ]; then
        echo unable to find Pgpool-II backend node id for $1:$2
        return 255
    else
        echo attaching node-id: $node_id to Pgpool-II
        attach_node_id $node_id
        return $?
    fi
}

# function detaches the backend node identified by hostname:port
# from Pgpool-II
# Arguments:
# 1- Hostname
# 2- Port (optional) if not provided, node-id of first matching
#           hostname will be promoted
#
function detach_node {
    get_pgpool_nodeid_from_host $1 $2
    node_id=$?
    if [ $node_id -eq 255 ]; then
        echo unable to find Pgpool-II backend node id for $1:$2
        return 255
    else
        echo detaching node-id: $node_id from Pgpool-II
        detach_node_id $node_id
        return $?
    fi
}

```

```

function print_usage {
    echo "usage:"
    echo "    $(basename $0) operation hostname [port]".
    echo "    operations:".
    echo "        check_primary: check if node has a primary
role".
    echo "        promote: promote node".
    echo "        attach: attach node".
    echo "        detach: detach node".
}

# script entry point
if [ -z "$1" ] || [ -z "$2" ]; then
    echo "ERROR: operation not provided"
    print_usage
    exit 1
fi
shopt -s nocasematch
case "$1" in
    "check_primary" )
        is_host_is_primary_pgpool_node $2 $3
        ;;
    "promote" ) echo "promote"
        promote_standby_to_master $2 $3
        ;;
    "attach" ) echo "attach"
        attach_node $2 $3;;
    "detach" ) echo "detach"
        detach_node $2 $3;;
    "watchdog" ) echo "detach"
        $PGPOOL_PATH/pcp_watchdog_info -w -U $PCP_USER -h $PCP_HOST
        -p $PCP_PORT -v;;
    *) echo "invalid operation $1".
        print_usage;;
esac
exit $?

```