ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 8, August 2014

Implementing Connection Pooling Without Specific Packages

Satpal Singh Kushwaha[1], Gurleen Virdi[2]

[1]Assistant Professor, Dept. of CSE, Modern Institute of Technology & Research Center, Alwar, Rajasthan, India

[2]M.Tech Scholar, Dept. of CSE, Modern Institute of Technology & Research Center, Alwar, Rajasthan, India

ABSTRACT Increased volume of users of the information has posed challenges in data retrieval as more the number of users, more is the volume of data and more concurrent need for data access. Hence, faster data retrieval is crucial for any application. Using traditional way for database connectivity is becoming a bottleneck. An efficient solution of DataBase Connection Pooling have been provided by various researchers and, hence, the database vendors. But all the solution till now have been provided for heavier applications where thousands of users need a database connection simultaneously. For such applications its perfectly feasible to use heavy packages and vendor specific drivers and APIs. But when we scale down the application to smaller organization like a college administration, a shop management application, a hospital management application that may have access to a few hundreds of users, but sufficient to chock the database server, these heavy package solution have some overhead. In this dissertation, we have tried to scale down this concept and trying to implement the same principal of connection pooling but without using vendor specific drivers or pooling specific packages.

KEYWORDS: database, connection pooling, scaling

I. INTRODUCTION

"Database" refers to the data themselves and supporting data structures. Databases are created to operate large quantities of information by inputting, storing, retrieving and managing that information[1]. With the advent of technology, the amount of information exchanged and saved has seen a tremendous increase. JDBC provides a standard library for accessing relational databases. Using the JDBC API, one can access a wide variety of different SQL databases with exactly the same Java syntax. There are seven standard steps in querying databases[2]: 1. Load the JDBC driver 2. Define the connection URL 3. Establish the connection 4. Create a statement object 5. Execute a query or update 6. Process the results 7. Close the connection

Opening a connection to a database is a time-consuming process. For short queries, it can take much longer to open the connection than to perform the actual database retrieval. Consequently, it makes sense to reuse Connection Objects in applications that connect repeatedly to the same database. And here comes the need and importance of concept of Connection Pooling.

"Connection pooling" spreads the connection overhead across several user requests, thereby conserving resources for future requests. It can improve the response time of any application that requires connections, especially Web-based applications. But if we are talking of small number of users but sufficiently large to throttle the database servers, then we need to find the solution. This is so because we have to be dependent on vendor specific user, which finishes the "single interface multiple use" concept of Java applications. Also, the packages that are used to implement this Connection Pooling mechanism create overhead in memory for small scaled applications.

II. RELATED WORK

Basic idea of Connection Pooling technology is to pre-establish some connection objects and place them in memory for use, when needing to establish database connection in program only to take one to use from memory without having a new, after using only need to replace them in memory, and the establishment and disconnection of linking are all managed by the connection pool itself[3]. A connection pool class should be able to perform the following tasks:

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 8, August 2014

1. Preallocate the connections: Allocating more connections in advance speeds things up if there will be many concurrent requests later but causes an initial delay.
2. Manage available connections: If a connection is required and an idle connection is available, put it in the list of busy connections and then return it.
3. Allocate new connections: If a connection is required, there is no idle connection available, and the connection limit has not been reached, then start a background thread to allocate a new connection. Then, wait for the first available connection, whether or not it is the newly allocated one.
4. Wait for a connection to become available: This situation occurs when there is no idle connection and you've reached the limit on the number of connections. This waiting should be accomplished without continual polling.
5. Close connections when required: Note that connections are closed when they are garbage collected, so you don't always have to close them explicitly. But, you sometimes want more explicit control over the process[2].

The configuration parameters[3] of centralized connection pool include:
1. *Min-Connections* established by connection pool, namely the idle connections maintaining dynamic connection pool;
2. *Max-Connections* in connection pool;
3. Connections maintaining dynamic connection pool, and in which
   Max-Connections = Min-Connections + Connections;
4. *Wait-Connection-Times* without idle connection;
5. Connection-Use-Count;
6. Wait-Release-Time.

c3p0 package[4] is an easy-to-use library for making traditional JDBC drivers "enterprise-ready" by augmenting them with functionality defined by the jdbc3 spec and the optional extensions to jdbc2. c3p0 hopes to provide DataSource implementations more than suitable for use by high-volume "J2EE enterprise applications".

The *"commons-dbcp" package*[5] by Apache relies on code in the commons-pool package to provide the underlying object pool mechanisms that it utilizes. DBCP 2 is based on Commons Pool 2 and provides increased performance, JMX support as well as numerous other new features compared to DBCP 1.x.

Database Resident Connection Pooling (DRCP) by Oracle[6] pools "dedicated" servers. A pooled server is the equivalent of a server foreground process and a database session combined. DRCP complements middle-tier connection pools that share connections between threads in a middle-tier process. In addition, DRCP enables sharing of database connections across middle-tier processes on the same middle-tier host and even across middle-tier hosts.

The *Java Naming and Directory Interface (JNDI)* is a Java API[7] for a directory service that allows Java software clients to discover and look up data and objects via a name. Like all Java APIs that interface with host systems, JNDI is independent of the underlying implementation. Additionally, it specifies a Service Provider Interface (SPI) that allows directory service implementations to be plugged into the framework. It may make use of a server, a flat file, or a database; the choice is up to the vendor.

JNDI with JDBC : JDBC is java database API, while JNDI is java naming and directory Interface API. The main thing in here is that in a JNDI directory you're actually storing a JDBC DataSource, so, you're simply using JDBC and obtain a Connection via JNDI lookup[8]. In short words: JDBC is Database realm, JNDI lets you store Objects in a virtual context (the Directory) that can be local, remote (Implementation details usually don't matters).

Data Sources: The JDBC 2.0 extension API introduced the concept of *data sources*, which are standard, general-use objects for specifying databases or other resources to use. Data sources can optionally be bound to Java Naming and Directory Interface (JNDI) entities so that you can access databases by logical names, for convenience and portability[9].

DB2(R) Connect Enterprise Edition servers by IBM often provide database connections for thousands of simultaneous client requests. In DB2 Connect Version 6 and later connection pooling is activated by default[10].
When a DB2 Connect instance is started a pool of *coordinating agents* is created.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 8, August 2014

Step 1: When a connection request comes in an agent is assigned to this request. The agent will connect to the DB2 server and a thread will be created in DB2.

Step 2: When the application issues disconnect request, the agent will not pass this request along to the DB2 server. Instead, the agent is put back in to the pool. The agent in the pool still owns its connection to the DB2 server and the corresponding DB2 thread.

Step 3: When another application issues a connect request, this agent is assigned to this new application. To ensure secure operation, user identity information is passed along to the DB2 thread which in turn performs user authentication.

Step 4: When an application requests disconnection from the host, DB2 Connect drops the inbound connection with the application, but keeps the outbound connection to the host in a pool.

The connection pooling provided by *OCI Driver* in Oracle9*i*[9] allows applications to have many logical connections, all using a small set of physical connections. Each call on this logical connection will be routed on the physical connection that is available at that time. Call-duration based pooling of connections is a more scalable connection pooling solution.

III. PROPOSED ALGORITHM

A. *Design Considerations:*

- Front End(Programing env.) : Java 5 (jdk1.5 and jre 1.5) or above
- IDE : Eclipse Java EE IDE for Web Developers; Juno Service Release 2
- Back End(DataBase) : MySQL 6.0
- Application Server : JBoss 5.0 (which is built above Apache Tomcat)

B. *Description of the Proposed Algorithm:*

We are going to use simple data structures provided by Java 5 and onwards for concurrent applications specifically.
1. *Executer:* One of the important interface that we have in this framework is the Executer interface. It is simply an object that executes runnable tasks. It decouples task submission from the details of how a task will be executed.

2. *ConcurrentLinkedQueue:* An unbounded thread-safe queue based on linked nodes. This queue orders elements in FIFO (first-in-first-out). It is an appropriate choice when many threads will share access to a common collection.

3. *ScheduledExecutorService:* is an ExecutorService which can schedule tasks to run after a delay, or to execute repeatedly with a fixed interval of time in between each execution. Tasks are executed asynchronously by a worker thread, and not by the thread handing the task to the ScheduledExecutorService.

The basic parameters we are using to handle the connection pooling operations are:
1. MIN_IDLE : minimum number of idle connection objects to be present in the pool
2. MAX_CON : maximum number of connection objects in the pooling system. This includes both idle as well as active connections.
3. WAIT_TIME: time interval after which a separate thread that checks for these two constraints to be met.

We have used 2 queues to implement the pool collectively:

(i) IDLE queue: which is going to hold the ON but idle connection objects. Idle means the connection is not involved in any transaction and is available.
(ii) ACTIVE queue : which will be holding the ON and active connection objects. Active here means the connection is busy with certain transaction.

1. PoolCreation : a method used to initialize the pool with MIN_IDLE number of objects and maintenance of MAX_CON objects in the system of queues

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 8, August 2014

2. BorrowObject : this method is used to allocate the connection object from the pool when requested for.
3. ReturnObject : this method is used to return the object to the pool when asked for it.
4. CreateObject : this is one of the important method that actually do the work of connection object creation and database connection establishment.
Global variable : POOLSIZE = no. of objects in IDLE + no. of objects in ACTIVE

IV. PSEUDO CODE

Algorithm 1 PoolCreation(MIN_IDLE,MAX_CON,WAIT_TIME) :
(1) (a) Initialize the IDLE queue with MIN_IDLE number of connection objects. Use CreateObject method to create an individual connection object and make it live and ready for use.
(b) POOLSIZE-=MIN_IDLE
(2) With the initialization of pool, schedule a thread in parallel which is going to keep a check on number of objects every WAIT_TIME seconds.
If (POOLSIZE > MAX_CON) then
While (IDLE > 0)
(i) remove the object from IDLE
(ii) reduce POOLSIZE by one as an atomic operation

Algorithm 2 : BorrowObject :
(1) If (IDLE queue is empty and POOLSIZE<MAX_CON) then
(i) Create a connection object through *CreateObject* method
(ii) Increment the POOLSIZE as an atomic operation.
(2) if (IDLE is not empty) then
(i) Remove an object from IDLE queue and store in a variable.
(3) Add the object(either created in Step 1 or borrowed from Step 2) into the ACTIVE queue.
(4) Return the object taken.

Algorithm 3 : ReturnObject(OBJ) :
(1) Remove the specific connection object OBJ from ACTIVE
(2) Add this object, OBJ, into the IDLE

V. SIMULATION RESULTS

A Test Plan defines and provides a layout of how and what to test. It can be viewed as a container for running tests(test cases). Considering our webapp, the Online Shopping Portal, we have considered the most complicated test case for testing our mechanism viz. *"Registered User" comes in→browses through the site→makes purchase→checks out→finally finishes.* This fires a *fresh query at each step* for each user. So this test case *generates the most dense set of SQL queries* and hence most dense requirement of database connections in concurrent environment. We have performed the load testing for the considered test case through widely and freely available open source load testing tool "JMeter". The test case has been tested in 3 versions for a better comparison:

1. Non pooled version(*WOCP*, WithOut Connection Pool), where traditional method of database connectivity is used.
2. JNDI version(*WJNDI*, With JNDI), where connection pooling with JNDI is used, one of the simplest mechanism for pooling at large scale.
3. *Pooled* version of application, where our mechanism is integrated into the application.

Test Script Recording element is used to create the test script and the test case is run under varying load as depicted in Table 1

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 8, August 2014

Table 1:Test Case set for varying load on the application; for PoolCreation(5,15,4)

Run	#Threads	Ramp-Up time (sec)	Loop Count	Burst (req/sec)	Total virtual users (#Threads X Loop)
RUN1	10	2	10	5	100
RUN2	15	2	10	7.5	150
RUN3	20	2	10	10	200
RUN4	25	2	10	12.5	250
RUN5	30	2	10	15	300
RUN6	35	2	10	17.5	350
RUN7	40	2	10	20	400
RUN8	45	2	10	22.5	450
RUN9	50	2	10	25	500
RUN10	55	2	10	27.5	550
RUN11	60	2	10	30	600
RUN12	65	2	10	32.5	650
RUN13	70	2	10	35	700
RUN14	75	2	10	37.5	750
RUN15	80	2	10	40	800
RUN16	85	2	10	42.5	850

The corresponding JMeter results are collected(in condensed form) in Table 2 from the listeners used in test script.

Table 2 : Throughput for 3 different versions of the application for performance comparison

Sample Run	Throughput (in req/min) <WOCP version>	Throughput (in req/min) <WJNDI version>	Throughput (in req/min) <PoolCreation method>
RUN1	665.2132	891.123	962.371
RUN2	1116.863	1213.170	1282.015
RUN3	1326.63	1489.228	1573.266
RUN4	1779.45	1944.767	2058.146
RUN5	1736.53	1973.282	2108.448
RUN6	2101.681	2292.970	2287.368
RUN7	2410.271	2842.063	2982.289
RUN8	2743.753	3046.864	3181.869
RUN9	3054.003	3182.206	3310.623
RUN10		3574.075	3759.675
RUN11		4074.407	4221.477
RUN12		4197.094	4729.894
RUN13		4537.414	4709.512
RUN14		4881.526	5134.700
RUN15		5443.686	5555.951
RUN16		5674.203	5743.885

The final results can be visualized in the fig 1.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 8, August 2014

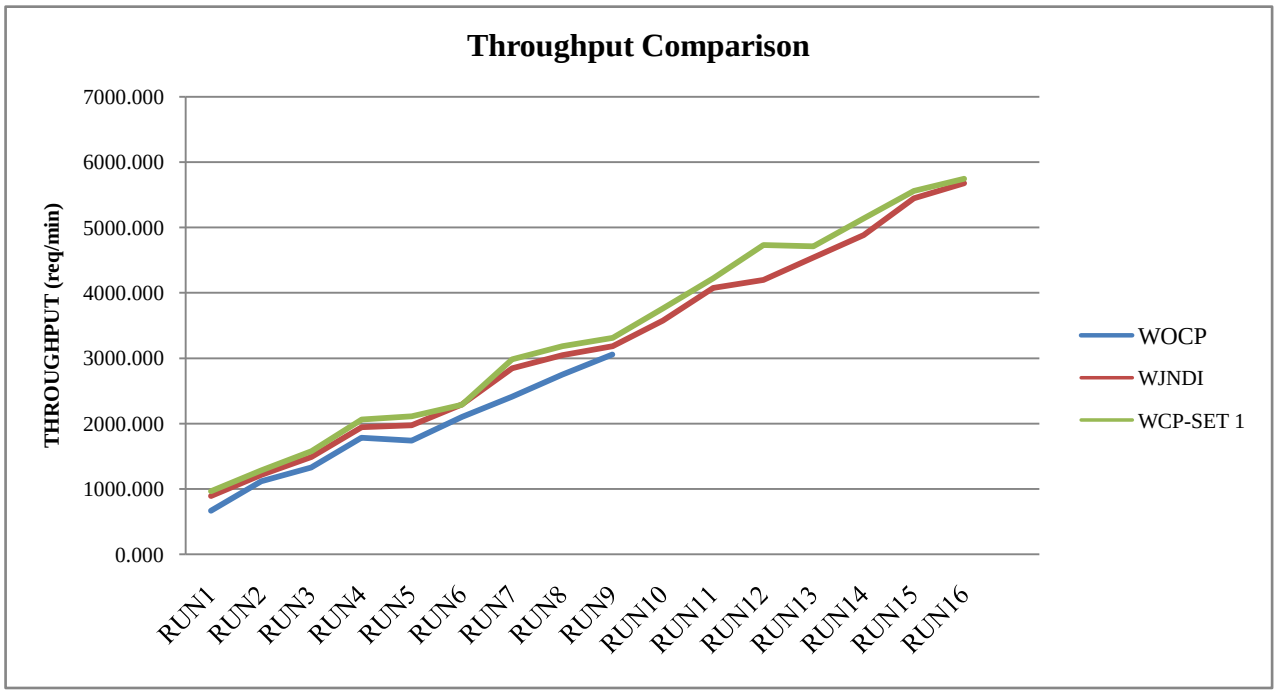

Fig 1 : Throughput comparison of 3 versions of application viz, non-pooled(WOCP),Pooled and JNDI version(WJNDI)

After calculating the respective ratios of "pooled to WOCP" throughput and "pooled to WJNDI" throughput, we came to the following results that

Pooled/WOCP=1.178 (Approx.) ➔ Pooled=1.178 X WOCP ➔ ~18 % performance increment (in throughput) over Non-pooled environment

Pooled/WJNDI=1.041 (Approx.) ➔ Pooled=1.041 X WJNDI ➔ ~4% performance increment (in throughput) over JNDI mechanism

This method of scaling down the application and implementing DataBase Connection Pooling not only performs better for smaller application, but also maintains the application architecture to simple 2-tier architecture. Tier-1 being the client side of the application while Tier-2 being the server side(including both Application Server and Database Server). While the available simplest method of pooling that we used, JNDI naming with connection pooling, breaks the application into 3-tier structure. Tier-1 is client, Tier-2 is application server and Tier-3 being the database server, hidden behind the JNDI naming in application server.

VI. CONCLUSION AND FUTURE WORK

The results showed that the proposed algorithm performs better with overall performance gains along with architectural benefits. JNDI breaks the application into 3-tier architecture while our proposed system maintains a simple 2-tier architecture of the application. It has a minor scope of improvement that it did not include the wait time for the requesting thread if the pool is fully occupied and there is no room for further adding a new connection object. This can be considered as a future work for this thesis work.

REFERENCES

1. Wikipedia, *the free encyclopedia*, Available : http://en.wikipedia.org/wiki/Database
2. http://pdf.coreservlets.com/first-edition/CSAJSP-Chapter18.pdf
3. Guo-liang Feng, and Lian-he Yang, "A New Method in Improving Database Connection Pool Model" *International Journal of Computer and Information Engineering, Vol 1, Issue 7 ,2007*
4. http://www.mchange.com/projects/c3p0/

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 8, August 2014

5. Apache Tomcat Project ; http://commons.apache.org/proper/commons-dbcp/
6. Krishna Mohan Itikarlapalli, Kevin Neel, Sreekumar Seshadri, Luxi Chidambaran, Amoghavarsha Ramappa, Srinath Krishnaswamy, Santanu Datta, Oracle Corporation, World Headquarters, *"Database Resident Connection Pooling (DRCP),Oracle Database 11g"*, Technical White paper, September 2007
7. Wikipedia, Available : http://en.wikipedia.org/wiki/Java_Naming_and_Directory_Interface
8. http://stackoverflow.com/questions/15676990/difference-of-connection-pool-jdbc-and-jndi
9. Oracle Documentation , Available : http://docs.oracle.com/cd/B10501_01/java.920/a96654/connpoca.htm
10. IBM Documentation : http://www01.ibm.com/support/knowledgecenter/SSEPGG_8.2.0/com.ibm.db2.udb.doc/conn/c0006170.htm?lang=en

BIOGRAPHY

Satpal Singh Kushwaha is an Assistant Professor in the Computer Science and Engineering Department, Modern Institute of Technology and Research Center, Alwar, Rajasthan, India which is affiliated to Rajasthan Technical University, Kota, Rajasthan, India. He completed his Master of Technology (M.Tech) degree in 2012 from Rajasthan Technical University, Kota, Rajasthan, India. His research interests are Deep Web, Hidden Web, Search Engines etc.

Gurleen Virdi is a Post Graduate Research Scholar in the Computer Science and Engineering Department, Modern Institue of Technology and Research Center, Alwar, Rajasthan, India which is affiliated to Rajasthan Technical University, Kota, Rajasthan, India. She completed her Engineering degree with Honours in 2007 from Laxmi Devi Institute of Engineering and Technology, Alwar, Rajasthan, India. Her research interests are DataBase, Logic Programing, Artificial Intelligence, etc.