JENNIFER Customer Case Study
(adapt to WebLogic)

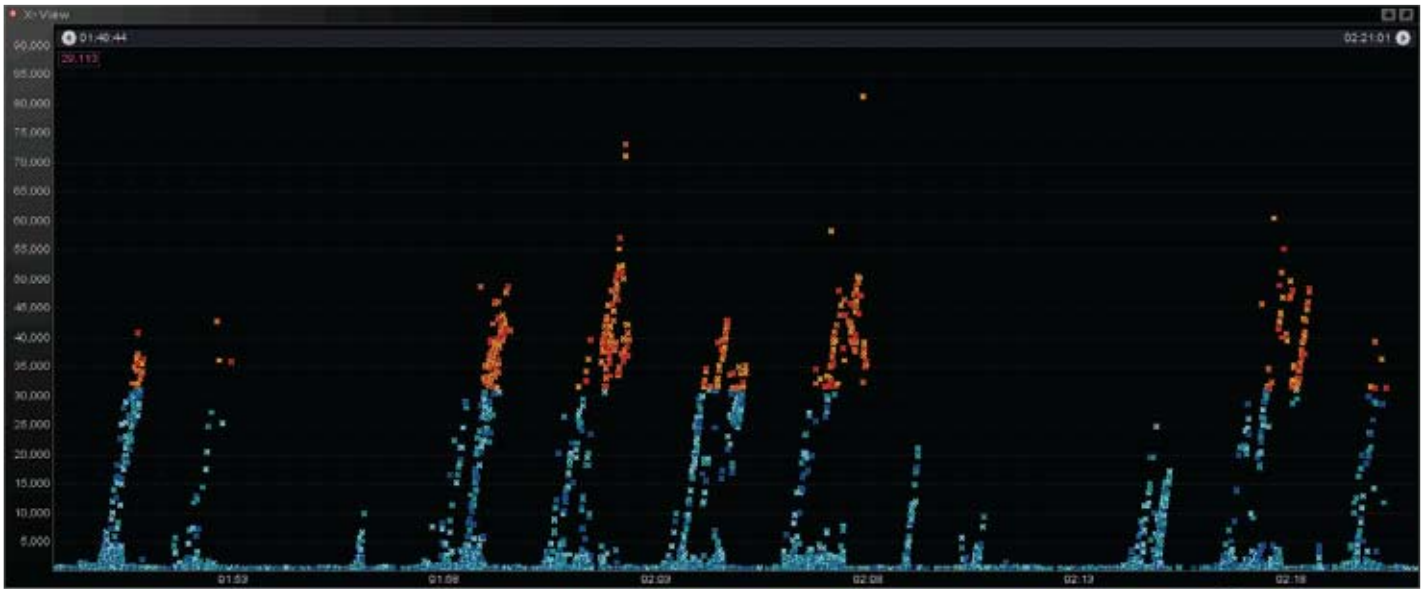
JENNIFER
JENNIFERSOFT, Inc.

This case study is an example of how one customer uses JENNIFER on WebLogic application server.

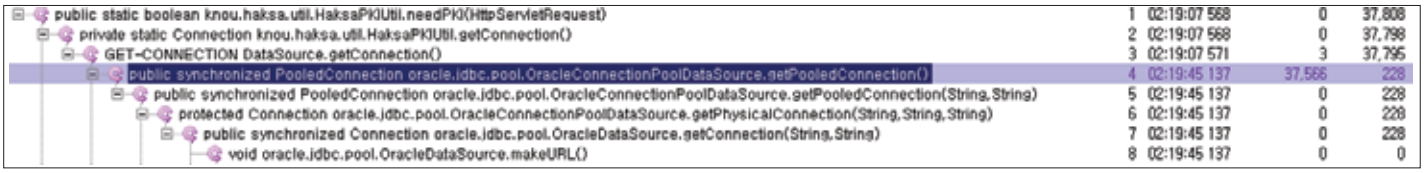
Title : Frequent JNDI Look Up causing delay in getting JDBC connection.

Summary : After 2 pm, when the system usage is highest, the response time of applications are being delayed. Customers have to wait for more than 1 minute after submitting requests; many customer gave up using the service because of long response time. This performance problem prevented the application from servicing the customers, thus negatively impacting business.

[Verify the transactions associated with delayed response time using X-View]



[Analyze the detailed profiling of application with delayed response time]



public static boolean knou.haksa.util.HaksaPKIUtil.needPKI(HttpServletRequest)	1	02:19:07 568	0	37,808
private static Connection knou.haksa.util.HaksaPKIUtil.getConnection()	2	02:19:07 568	0	37,798
GET-CONNECTION DataSource.getConnection()	3	02:19:07 571	3	37,795
public synchronized PooledConnection oracle.jdbc.pool.OracleConnectionPoolDataSource.getPooledConnection()	4	02:19:45 137	37,566	228
public synchronized PooledConnection oracle.jdbc.pool.OracleConnectionPoolDataSource.getPooledConnection(String, String)	5	02:19:45 137	0	228
protected Connection oracle.jdbc.pool.OracleConnectionPoolDataSource.getPhysicalConnection(String, String, String)	6	02:19:45 137	0	228
public synchronized Connection oracle.jdbc.pool.OracleDataSource.getConnection(String, String)	7	02:19:45 137	0	228
void oracle.jdbc.pool.OracleDataSource.makeURL()	8	02:19:45 137	0	0

Problem Analysis

```
public class HaksaPKIUtil
{

    public HaksaPKIUtil()
    {
    }

    private static Connection getConnection()
        throws NamingException, SQLException
    {
        DataSource datasource = (DataSource)(new InitialContext()).lookup("haksa_DS");
        return datasource.getConnection();
    }
```

After analyzing the profiling data of services with delayed response time, system engineers determined that the source for establishing JDBC Connection was written to run JNDI Look up everything a connection was established. Frequent JDNI Look up caused JDBC resource shortage.

Problem Fix

```
public class HaksaPKIUtil
{
private static DataSource datasource = null;


public HaksaPKIUtil()
{
}

private static Connection getConnection()
throws NamingException, SQLException
{
if ( datasource == null ) {
datasource = (DataSource)(new InitialContext()).lookup("haksa_DS");
}

return datasource.getConnection();
}
```

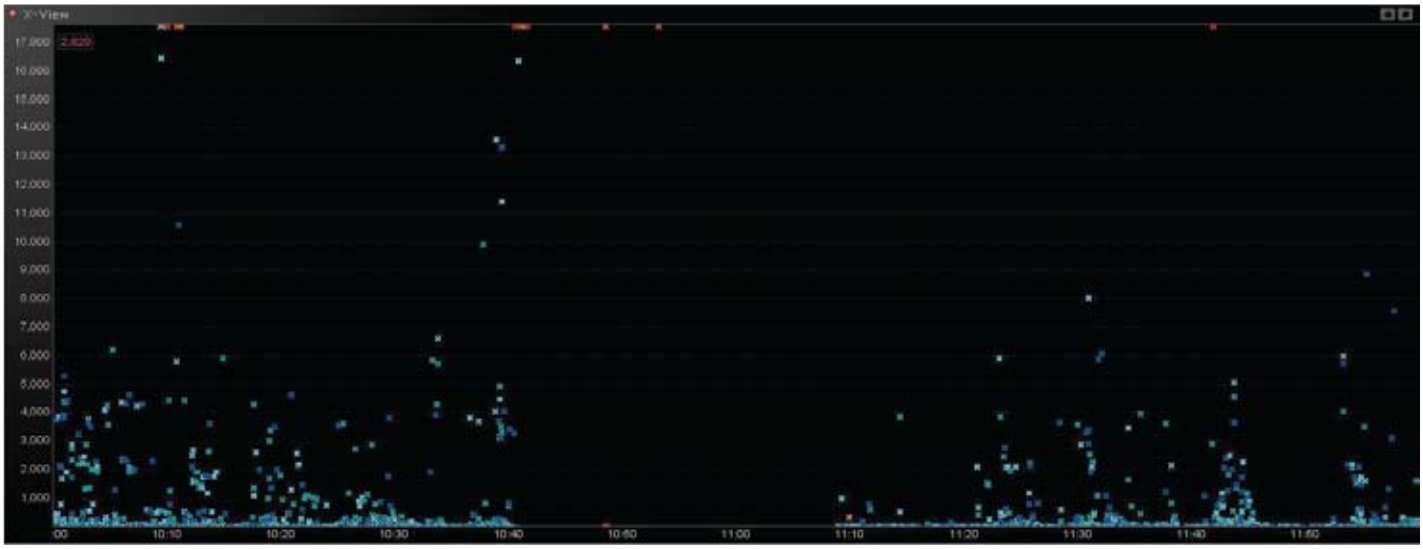
Code source was change to run JNDI Look up once only when system is initialized to eliminate the possibility of JNDI Look up causing JDBC resource problem.

JENNIFER Customer Case Study
(adapt to WebLogic)

JENNIFER
JENNIFERSOFT, Inc.

Title : OOM (Out of Memory) error due to excessive use of “fetch” command against Database.
Summary : Excessive use of database fetch causes Out of Memory error, resulting in System Hang.

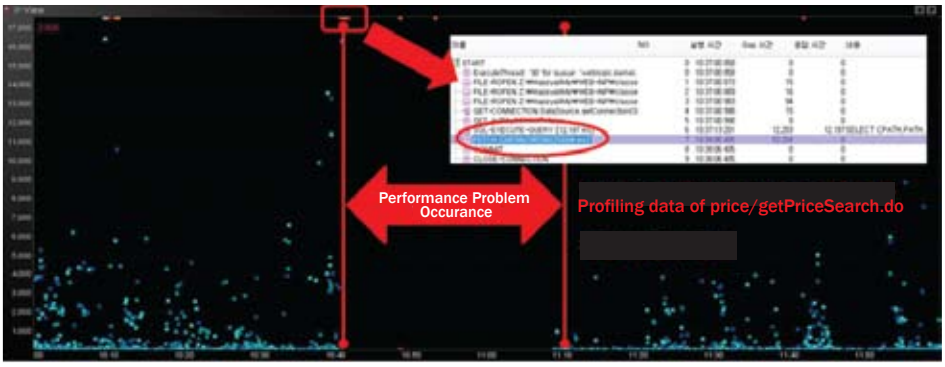
[Verify application down-time or “System Hang” using X-View]



Problem Analysis

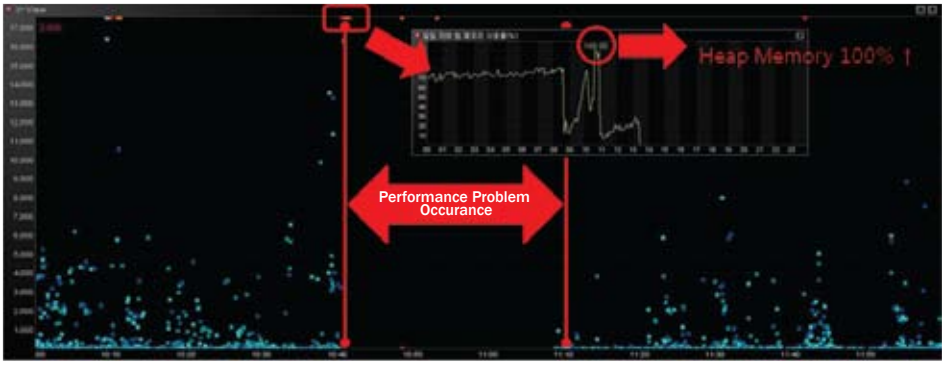
After analyzing the services with high response time that finished moments before system hang, system engineers found that the application (/price/getPriceSearch.do) submitted request for more than 24,000 DB fetch commands all at one time, causing shortage in heap memory and causing Out of Memory error.

[Application which was ran just before ((/price/getPriceSearch.do) Out of Memory(OOM) occurred)



Compare X-View with Heap Memory monitoring graph to see that heap memory usage was near 100% when the performance problem occurred.

[X-View and Heap Memory Usage when Out of Memory(OOM) error occurred]



Problem Fix

To resolve performance issue associated with processing large quantity of data, DB query was tuned to only grab data which is necessary for output to screen, thus getting enabling normal service again.