Splunk® DB Connect Deploy and Use
Splunk DB Connect 3.1.4

Architecture and performance considerations

Generated: 6/07/2019 2:59 pm
Architecture and performance considerations

When adding Splunk DB Connect to your deployment, there are several architecture and performance considerations to take into account. You can install and run Splunk DB Connect on Splunk Enterprise deployments ranging from a single host (indexer and Splunk Web both running on the same system) to a large distributed deployment (multiple search heads, search head clusters, indexers, load-balanced forwarders, and so on). This topic provides guidance for setting DB Connect up and running in these environments. It also describes the kind of performance you can expect based on your deployment and capacity requirements.

Database performance considerations

If Splunk DB Connect retrieves large amount of data from your database, it may affect your database performance, especially for the initial run. Subsequent runs of the same query may have less impact, as the database may cache results and only retrieve new data since the previous run of the query.

Performance considerations in distributed environments

To use Splunk DB Connect in a distributed search environment, including search head clusters, you must determine the planned use cases. For ad hoc, interactive usage of database connections by live users, install the app on search head(s). For scheduled indexing from databases and output of data to databases, install the app on heavy forwarder(s).

Note: DB Connect does not support running scheduled inputs and outputs on a search head cluster. Splunk recommends you run inputs and outputs from a heavy forwarder. Beginning with DB Connect 3, this recommendation is enforced. If you have been running scheduled inputs or outputs on a search head cluster using DB Connect 1 or DB Connect 2, you will need to move them onto a heavy forwarder before migrating to DB Connect 3.

When planning a large DB Connect deployment, the ideal configuration for your needs can depend on a number of factors, including:

- Total number of Forwarders in the deployment, and the hardware specifications of each.
- Total expected data volume to transfer.
• Number of database inputs per Forwarder.
• Dataset size, per input, per interval.
• **Execution Frequency**, the interval length between a database input’s separate executions.
• Fetch size (note that not all JDBC drivers use this parameter for returning result sets).

Overloading the system can lead to data loss, so performance measurement and tuning can be critical. Use performance expectations as the reference to plan your deployment, and monitor expected data returns for loss conditions.

**Performance expectations**

This section provides measured throughput data achieved under certain operating conditions. Use the information here as a basis for estimating and optimizing the DB Connect throughput performance in your own production environment. As performance may vary based on user characteristics, application usage, server configurations, and other factors, specific performance results cannot be guaranteed.

The performance data in the following table were produced with the following test bed and DB Connect configuration (note that increasing cores and/or RAM may improve scaling characteristics):

• Server: 8-core 2.60GHz CPU, 16GB RAM, 1Gb Ethernet NIC, 64bit Linux
• JVM config: MaxHeapSize = 4GB. (For more information about the JVM memory setting, see "Performance tuning advice".)
• Data Source: Oracle 11g

**Inputs**

• Number of inputs: 1600
• Data payload (per input execution) : 250KB
• Duration = 45 minutes
• Interval: 1 minute

\[
\text{total data volume} = \text{data payload} \times \text{duration} / \text{interval} \times \text{number of inputs} = 17.5 \text{ GB}
\]

Data payload per input execution is the same for different input modes (rising column and batch)
**Queries**

<table>
<thead>
<tr>
<th>Rows in data set</th>
<th>DB Connect 3</th>
<th>DB Connect 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.2 seconds</td>
<td>1.4 seconds</td>
</tr>
<tr>
<td></td>
<td>1.3 seconds</td>
<td>1.5 seconds</td>
</tr>
<tr>
<td></td>
<td>1.6 seconds</td>
<td>2.4 seconds</td>
</tr>
<tr>
<td></td>
<td>4.1 seconds</td>
<td>11.4 seconds</td>
</tr>
<tr>
<td></td>
<td>22.9 seconds</td>
<td>103.5 seconds</td>
</tr>
</tbody>
</table>

**Lookups**

<table>
<thead>
<tr>
<th>Rows in data set</th>
<th>DB Connect 3</th>
<th>DB Connect 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.2 seconds</td>
<td>0.2 seconds</td>
</tr>
<tr>
<td></td>
<td>2.8 seconds</td>
<td>4.3 seconds</td>
</tr>
<tr>
<td></td>
<td>36.0 seconds</td>
<td>70.0 seconds</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Rows in data set</th>
<th>DB Connect 3</th>
<th>DB Connect 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.1 seconds</td>
<td>1.0 seconds</td>
</tr>
<tr>
<td></td>
<td>1.9 seconds</td>
<td>1.5 seconds</td>
</tr>
<tr>
<td></td>
<td>3.0 seconds</td>
<td>10.0 seconds</td>
</tr>
<tr>
<td></td>
<td>9.1 seconds</td>
<td>83.9 seconds</td>
</tr>
<tr>
<td></td>
<td>67.2 seconds</td>
<td>644.0 seconds</td>
</tr>
</tbody>
</table>

**General performance tuning considerations**

While it's impossible to provide prescriptive advice for maximizing performance in every situation, the following observations and tips can help you tune and improve performance in your unique distributed deployment:

1. **Only select columns if you really need them.** A table can contain many types of columns. When ingesting data from a database into DB Connect, you likely don't need all of them. Therefore, instead of using a `SELECT * FROM ...` clause to fetch all the columns, select only what you need by using a `SELECT columnNeeded1, columnNeeded2, ... FROM ...` clause. More columns means more memory claimed by the task server; omit those unnecessary columns to make smarter use of your available memory. See SQL tips and tricks for more details.

2. **Avoid reaching the 5MB/10MB limit.** Very large column sizes can cause DB Connect to potentially run out of memory and behave erratically, so DB Connect has a column size limit of 10MB for data columns that hold two-byte data types and 5MB for one-byte data types. Columns with data
exceeding these limits will have their data truncated. If possible, trim the
amount of data stored per column so that you avoid the DB Connect hard
caps.

3. **Adjust the fetch size based on your scenario.** The **Fetch Size** input
parameter specifies the number of rows returned at a time from a
database, which defaults to 300 rows. A higher fetch size means more
records are received per database request, so fewer database requests
are required to retrieve the same total number of records. This increases
resource utilization on the database and in DB Connect, but can lead to
performance improvements. Lowering the fetch size parameter can help
prevent the Task Server from hitting its memory limit. If you receive out of
memory errors when you increase the fetch size, you may need to
increase the memory heap size from its default of 1/4 of system RAM.

4. **Reduce the total number of database inputs.** It can increase the
amount of data that each input is taking in. This helps ensure that CPU
cores have to handle fewer processes within a given window of time.
Small datasets can be slower than large because of environment
initialization.

5. **Reduce the concurrency of scheduled database tasks.** Shifting start
times for scheduled tasks will reduce choke points during which inputs and
outputs have to share resources. For more information, see "Set
parameters" in Create and manage database inputs.

6. **Adjust batch_upload_size field.** The **batch_upload_size** field defines
the number of events sent to splunkd through HEC per request, which
defaults to 1,000 records. A higher batch upload size means more records
are sent per HTTP post, so fewer server transactions are required to index
the same total number of records. This increases resource utilization on
the Forwarder, but can lead to performance improvements. You can
increase the **batch_upload_size field** under

```bash
$SPLUNK_HOME/etc/apps/splunk_app_db_connect/local/db_inputs.conf
```
to have better performance.

7. **Specify sufficient hardware.** In general, Splunk recommends the same
hardware specifications for DB Connect as it does for Splunk Enterprise.
Increased hardware may be necessary for increased indexing loads.

8. **Configure Java for performance.** Current Java engines will
automatically reserve 25% of the machine’s RAM on launch. If your **JVM
Options** setting is specified as `-Xmx1024m` (which is the default value
from DB Connect V2.0 to v2.2). You can remove it and use the default
JVM setting. For more information about changing JVM options, see "JVM
Options" in Configure DB Connect Settings.

9. **Configure Splunk for performance.** Increase Splunkd’s index queue
size and number of Parallel Ingestion Pipelines to avoid concurrency
limits.
10. **Configure DB Connect for performance** Set SchedulerThreadpoolSize to match the number of processor cores.

**Performance non-factors**

During testing, varying the following factors had a negligible effect on performance:

- There was no discernable performance difference between running in batch mode (all events processed) and running in rising column mode (just the new events processed) with the same dataset.
- The number of defined database connections does not limit performance. Note that the number of connections is different from the number of database inputs.

**More performance help**

If you are still experiencing performance issues, or want to receive feedback tailored to your setup, you have the following options:

- Post a request to the community on Splunk Answers.
- Contact Splunk Support.