P6 EPPM Job Services
Performance Tuning and Settings

ORACLE
PRIMAVERA

Disclaimer

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Overview

The document will explain how administrators can set up P6 Job Services to handle jobs efficiently and scale to the degree that is needed in their server environment. This includes memory considerations, concurrency (threading) options, and multiple job service deployment options. This document is applicable to P6 EPPM versions 8.1 and above.

Introduction

P6 Job Services are designed to allow flexibility for running P6 jobs in a distributed system.

P6 Job Services include the following major services that can be scheduled or manually invoked:

- Project Scheduler (Project Scheduled Services)
- Leveler (Project Scheduled Services)
- Summarizer (Project Scheduled Services)
- Apply Actuals (Project Scheduled Services)
- Publish Enterprise Security, Resource, and Enterprise summary data (Global Scheduled Services)

The following additional job services can be invoked through user actions:

- Project Publication
- Store Period Performance
- Recalculate Assignment Cost
- Send To Fusion
- Sync Actuals/Remaining/Planned units

What is a P6 Job Service Job?

A P6 job is a major functional procedure that performs some sort of transformation against the P6 database. The job can run independently on any server that is set up to run P6 Job Services. The job inputs are described in the job record within the JOBSVC table. The job output is usually a transformation against the database (such as date recalculations for a project scheduling job). The JOBSVC record for the job also contains a status code which describes the status of the job (pending, running, complete, or failed). P6 jobs are created by P6 users when they click items such as "Schedule Project" in the web GUI. P6 jobs can also be scheduled to run automatically on a daily, weekly, or

monthly basis. The scheduled "recurring" jobs are configured in the Project Scheduled Services and Global Scheduled Services dialogs in P6.

What is a P6 job server?

A P6 job server is a server component that runs P6 jobs. It will look for pending jobs, run them, then set their job status to reflect the success/failure status of the job. A job server uses the parameters that were stored in the job record in order to carry out the process.

The P6 Job Server functionality is built in to a number of P6 modules, including:

- p6services.jar (Primavera Job Server)
- p6.ear (P6 Web application)
- p6ws.ear (P6 Web Services)
- intgserver.ear (P6 Integration API)
- Primavera API.ear (Primavera Integration API Server)

Only p6services.jar was built to handle P6 jobs exclusively, and is a very easy deployment since it is a simple Java application requiring no middleware application server components. The other modules also contain the same P6 job server functionality, but they must share resources with other their primary tasks such as serving web requests or API/Web Services calls.

P6services Terminology

When this document refers to p6services or P6 Job Server, it is generally referring to the p6services.jar module. However since p6.ear and other modules also contain the same p6services functionality, the same rules apply for those modules as well.

Enabling job processing for various types of jobs

The Primavera P6 Administrator tool allows you to configure job service settings, including which types of jobs will be processed, number of concurrent threads, and job polling rate. Each P6 module (P6 Web, p6services, API, etc) will obey the job service settings in the configuration which is being used by that module. All of the Job Services settings are under the {ConfigurationName}/Services node in the P6 Administrator tool.

Setting up P6 Job Services to allow job distribution across multiple machines

It is very simple to deploy p6services.jar on additional hardware in order to distribute the job load. Adding new p6services deployments to the system is simply a matter of installing the p6services.jar, configuring it to point to the P6 configuration database, then starting up the JVM process with recommended memory parameters. Please see the **P6 EPPM Post Install Admin Guide** for detail on p6services.jar installation. Each deployed p6services process automatically polls the database for new jobs of the types that it is allowed to process, transfers the job(s) to a worker thread, and begins processing. By default each job type has 2 worker threads. This worker thread count is controlled by the **Concurrent Threads** setting for each job service. For example, if a p6services process has its *Scheduler*/Concurrent Threads setting set to 2 and its *Summarizer*/Concurrent Threads setting set to 4, then for any configured P6 database instance, at most 2 scheduler jobs and 4 summarizer jobs can be processed concurrently by that JVM. If the P6 Job Services configuration contains multiple database instances, the p6services deployment would use 2 scheduler and 4 summarizer threads *per database instance*. This is important to note when considering CPU and memory availability.

IMPORTANT:

The Concurrent Threads setting for the various services will apply *to each database instance* that is configured. So if Concurrent Threads for a service is 2, but there are 2 database instances, then there could potentially be 4 threads running that service (2 for each db instance). Because of this behavior, it is recommended that you always only have a single database instance per P6 services configuration. Having multiple database instances could cause unintentional high CPU usage and could negatively impact performance.

Using single or multiple P6 Services configurations

If p6services is deployed multiple times, then you will need to decide whether each p6services should either share the same configuration, point to their own configuration, or some combination in between.

The important factors to consider are:

-Which job service types you want to enable for each p6services instance?

-How many threads for each job service do you need to enable for each p6services instance?

For example, let's say you want 2 identical p6services deployments on 2 machines (2 machines each deploying a single instance of p6services). Both machines' hardware is identical, and you want both to handle the exact same types of jobs. In this case it would probably make sense to have both p6services deployments share the same configuration. That will ensure that the services thread counts and other

settings are always the same. So you create a configuration called "p6services_alpha" which both p6services deployments would reference.

Then let's say later you want to add a third machine that exclusively handles Project Publication jobs, and it has 6 CPU cores which is more than the other machines. So you would create another configuration called "p6services_projpublish" and let this machine use that configuration, and you would set the Concurrent Threads to 6 for the Publish Project job type, and set Concurrent Threads for all other job types to 0, since you don't want this machine handling anything other than project publication jobs. You would then go back to the p6services_alpha configuration and set the Publish Project threads to 0.

CPU and thread requirements for p6services

Due to the potential heavy CPU utilization of P6 Job Services, it is recommended to set up job servers so that they are on different machines than the user-transactional servers like P6 web application. This ensures that the users' web experience is never compromised. Because job services are fairly CPU intensive, **the goal is to have no more than 1 job thread running per CPU core**. So if your hardware has 4 cores, you should attempt to configure the concurrent threads of the various job types so that you never have more than 4 jobs running concurrently on that machine. Please see the *Example Scenario* section below for more details.

Memory requirements for p6services.jar

Summarizer, Scheduler, Leveler, and Project Publication job services: allocate an additional 2 GB of heap memory (-Xmx) per active job services thread.

All other job services: allocate an extra 1 GB per each 4 threads (with a minimum allocation of 1GB)

Please note that these memory recommendations are starting points only. If running jobs from p6.ear within an application server, you should add the additional memory allocation requirements above to the original base heap allocation you have already allocated based on your concurrent user count. It is very possible you will be able lower the allocated memory after doing some monitoring. You should monitor JVM memory usage over a typical high load period to determine whether your current max heap settings are appropriate. See later section entitled *Determining more detailed memory usage information for P6 Job Services* for more information on memory monitoring.

Determining more detailed memory usage information for P6 Job Services

In order to determine more closely how much memory is needed by P6 Job Services threads for a given machine, it is necessary to do some test runs of your services such that all job worker threads will be utilized. These test runs should be performed by running all of the jobs that you expect to be run in your production environment over a period of time, causing all services threads to be busy. During this time, you will need to monitor the maximum Java heap memory that was used during the test period. When starting p6services for these test runs, you should allocate the maximum amount of heap memory possible (-Xmx) so that the JVM will never be starved for memory during the test run.

You can then use the JDK tool **jvisualvm** to monitor the heap usage over time. The CPU and Memory graphs on the Monitor tab in jvisualvm will show you a graph over time of CPU usage and Used Heap. Those 2 values provide the most meaningful information for helping determine how efficiently your job services process is performing. Turn off the Classes and Threads graphs since those values are unimportant.

JVisualVM resides in the JDK/bin folder. Start up jvisualvm on the target machine you want to measure. Double click on the p6services process to open it. For p6services.jar, the process will be called p6services.jar. For P6 Web application server, the process is called weblogic.Server. If there are multiple weblogic.Server entries, you will need to determine which process is running your job services. You can use the command line "jps -v" (Sun JVM) or jrcmd (JRockit JVM) to look at Java command line parameters to help figure out which process id is the one which is running p6.ear.

Once jvisualvm has opened the process, go to the Monitor tab and turn off the Classes and Threads graphs. Now run your test by creating your jobs. As a sanity check, confirm that CPU usage goes up when jobs are executing.

When jobs are finished running (CPU usage is low and stable), check Heap graph and look for the maximum value of Used Heap. Add another 250MB to that, and that should be the maximum heap value that you should use in the Xmx and Xms parameters for starting up your p6services process.

It is also recommended that you use the following additional JVM memory parameters when starting up p6services.jar with Sun JDK:

```
-XX:PermSize=128m -XX:MaxPermSize=256m -XX:NewSize=128m -
XX:MaxNewSize=256m -XX:SurvivorRatio=8
```

These settings allow class loading and new memory allocations to be optimal with respect to the resource needs of p6services.jar.

Example syntax for launching p6services.jar (in current folder) with max heap memory set to 4GB (bootstrap file located in C:\p6services):

```
java –Xms4096m -Xmx4096m -
Dprimavera.bootstrap.home=C:\p6services -XX:PermSize=128m -
```

```
XX:MaxPermSize=256m -XX:NewSize=128m -XX:MaxNewSize=256m -
XX:SurvivorRatio=8  -jar p6services.jar
```

Another tool that can be used to monitor the maximum memory usage over time is JRockit Mission Control (jmc/jrmc).

Example Scenario: Heavy scheduling and summarization with 200 concurrent users using P6 Web

In this example scenario, P6 Web is being used heavily by a maximum of 200 concurrent users. It was determined by the administrator that scheduling and summarizing were the most actively used job functions, followed closely by project publication. The remaining job types were rarely used.

Every night it is required to run scheduling and summarization of all the projects in 2 large EPS nodes (roughly 50 projects per EPS). 10 projects are then re-published early each morning. During the daytime, there are roughly 20 project schedules recalculated and 10 summarizations performed on demand by various users, mostly right before lunch.

Hardware: Total of 4 machines: p6services_1, p6services_2, p6web_1, p6web_2.

Each machine has 4 CPU cores

Details for each machine:

p6services_1 (running p6services.jar):

- CPU cores: 4
- Installed memory: 10GB
- JVM max heap (-Xmx) = 8192m
- Summarizer Concurrent Threads: 2
- Scheduler Concurrent Threads: 2
- All other job types: 0 threads

p6services_2 (running p6services.jar):

- CPU cores: 4
- Installed memory: 10GB
- JVM max heap (-Xmx) = 8192m

Summarizer Concurrent Threads: 2

Publish Project Concurrent Threads: 2

All other job types: 0 threads

p6web_1 (running p6.ear):

CPU cores: 4

Installed memory: 8GB

CPU cores: 4

JVM max heap (-Xmx) = 6144m

Summarizer, Scheduler, and Publish Project threads set to 0

All other job types: 1 thread each

p6web_2 (running p6.ear)

CPU cores: 4

Installed memory: 8GB

JVM max heap (-Xmx) = 6144m

Summarizer, Scheduler, and Publish Project threads set to 0

All other job types: 1 thread each

In the example above, the 2 p6web machines are load balanced. Since both summarization and scheduling are being done concurrently overnight on p6services_1 and _2, with each machine having 4 CPU cores, the summarizer thread count is set to 2 and the scheduler thread count is set to 2 for each of those machines. If it were known that those two types of jobs didn't generally overlap, then it would be probably be ok to set both summarizer and scheduler threads to 4 each.

Since p6services_1 and p6services_2 would each be running 4 jobs concurrently for a long period of time overnight, and they are scheduler and summarize jobs which generally require 2GB per thread, each JVM requires 8GB of max heap (-Xmx8192). Because of OS overhead, it is recommended to have an extra 1.5 to 2GB of memory installed, which is why those machines were provisioned with 10GB. For performance reasons it is very important that OS virtual memory not be used to satisfy JVM memory requirements.

The p6web_1 and p6web_2 machines are primarily being used for P6 Web application traffic. However, 1 thread has been enabled for each of the other job service types to allow those jobs to be processed and distributed across those 2 machines. It is important that the Summarizer, Scheduler,

and Publish Project concurrent thread counts be set to 0 on those configurations, so that they don't start processing those jobs that were meant for the p6services_1 and _2 boxes.

Behavior when manually invoking scheduler and summarizer from P6 Web application

Invoking jobs manually from Scheduled Services view

If P6 Web application has at least 1 concurrent thread enabled for scheduler or summarizer, then right clicking on one of these jobs in the Scheduled Services view and choosing Run Service will force that job to be run immediately *on the P6 application server where your browser is currently logged in.* The job will not be distributed onto any other server. Invoking jobs in this way has a drawback in that the invocation will fail if P6 Web application's job service threads are all busy handling other jobs.

However, if P6 Web does not have any threads enabled for scheduler or summarizer (or Enable All Services settings is set to false), then right clicking and choosing Run Service will create a **copy** of the existing recurring job so that it can be run by the first available p6services thread from *any* machine. The newly created job can be monitored in the Service Status window (available in Activity view).

Invoking jobs from Activity View

The activity view allows a number of P6 jobs to be invoked from the Actions, Run menu as well as the toolbar.

The behavior of invoking the Scheduler from the Activity View is described below. After clicking Run Service the following behaviors apply:

If P6 Web has services enabled and the scheduling service has at least 1 thread configured:

A JOBSVC record is created. If a thread is available on P6 Web appserver, the job will immediately be run there. If the schedule job takes longer than 20 seconds to complete (or 4 minutes to complete for Summarizer), the user is informed that the job will be completed in the background. The newly created job can then be monitored in the Service Status window (Actions, Service Status…)

If P6 Web has services disabled OR scheduling service has 0 threads configured OR if threads are configured but all busy handling other jobs:

A JOBSVC record is created and the user is informed that the process will require additional time to complete. An available p6service will run the job at earliest opportunity. The newly created job can be monitored in the Service Status window.

P6 Administrator Settings that affect Job Services performance

Services/{ServiceName}/Concurrent Threads – As mentioned above, each major job service has a Concurrent Threads setting that allows you to specify how many Java threads will be used for running jobs. For example, if you set Scheduler Concurrent Threads to 2, any p6services process that is using that configuration will be able to run 2 scheduler jobs (per database) simultaneously. You should adjust the Concurrent Threads for your services such that you will generally not have more jobs running concurrently than the number of CPU cores available to the JVM.

Database/Connection Pool [Long Running]/Maximum Lease Duration – Job services use the Long Running database connection pool for most operations against the database. The Maximum Lease Duration is the maximum amount of time that a database connection can be involved in a transaction before the connection is aborted and the data is rolled back. In general, the default value of 15 minutes is suitable for Scheduler and Summarizer, since those jobs are made up of smaller transactions, each of which usually don't exceed 15 minutes. However, if you find that job services are failing with log entries related to "Connection has been recycled", you should increase the Maximum Lease Duration setting for the Long Running connection pool.

Thread Pool/Maximum Long Running Task Duration – All job services use this setting to determine how long to wait for a single job to complete before attempting to terminate it. If a single job takes longer than the Maximum Long Running Task Duration, then that job thread *may* be interrupted in order to terminate the job and set its status to FAILED. However, there is no guarantee that the job can be effectively interrupted, as it depends on the type of operation being performed when the MaximumLongRunningTaskDuration limit is reached. When terminated in this way, the job error will be shown as failing with an InterrruptedException. In practice, it is very unlikely that a job will be terminated due to it exceeding the MaximumLongRunningTaskDuration.

NOTE: To ensure efficient cleanup of database resources, please make sure that the Maximum Long Running Task Duration is equal to or greater than the db Connection Pool [Long Running]/MaximumLeaseDuration.

Monitoring job status at database level

The JOBSVC table within the PM database maintains the state of all job service jobs. Each record in the JOBSVC table represents a single job, whether it is a scheduled recurring job or a one-time only job.

The important columns of the JOBSVC table are described in Appendix 1 at the end of this document.

In order to look at the overall completion statistics (Pending, Running, Complete, or Failure) of all jobs in the system, the following database query can be run (from PRIVUSER account):

select job_type, status_code,count(status_code) count from jobsvc group by job_type,status_code order by job_type

Some frequently asked questions:

Q: How do I determine which job services machine has processed (or is processing) a particular job?

A: If a job is in the running or completed state, the worker_host field of the job record will indicate the machine name that ran (or is running) the job. Additionally the machine name *may* be followed by ::<processid>, where <processid> is the process id of the java process.

Q: How do I determine how long it took to process a particular job?

A: Unfortunately there is currently no way query the database for job run durations, this must be obtained from the P6 web application on a per job basis. For *scheduled* summarizer and scheduler jobs in Project Scheduled Services, you can click the View Log File icon from Project Scheduled Services or which will display a summary of the last completed run. This will show the job start and completion timestamps.

For summarizer and scheduler jobs that have been generated manually in P6 by choosing Summarize or Schedule Projects from menu, you can view the start and completion timestamps by going into the Service Status view from activity view, and click the View Log File icon.

Future versions of P6 database schema will contain a JOBSVC column which records the jobs' start date/time, which can then be subtracted from last_run_date in order to obtain true job duration.

Possible Job Services Errors and Solutions

The following errors may be generated by Job Services either in the P6WebAccess logfiles or the specific job log, depending on the type of job.

#

Error: "Could not get JDBC Connection; nested exception is java.sql.SQLException: Lease request timeout while waiting for connection to become available"

(Doc ID 1497061.1)

This error can occur when a job service thread is attempting to obtain a database connection in order to perform a read or update operation, and there are currently no database connections available for a period of 30 seconds or more. This can happen when there are many other jobs or database operations being performed, thus preventing a database connection (from the long running db pool) from freeing up. In order to resolve this, you should try increasing the Maximum Connections setting for the Long Running connection pool (under database instance) within the P6 admin tool. The default is 50, try increasing it to 100. Please note: increasing this value beyond 50 is really just a stop-gap measure, you should attempt to determine why there are 50 or more concurrent database connections being used in the first place. This usually is a result of extremely slow database performance causing a backlog of requests vying for database connections.

Error : "ORA-01013: user requested cancel of current operation"

This error can occur if the duration of a job's database access exceeds the Maximum Lease Duration setting (as set in admin settings under database instance/Connection Pool [Long Running]). The default value is 15 minutes, but it is possible that summary operations or project publication jobs could exceed that duration if the project is large and contains much detail (many WBS and/or activities). If you get this error, try increasing the Maximum Lease Duration for the Long Running connection pool to 30 minutes and repeat the operation to see if the problem is resolved. If error is occurring from Project Publication and you don't want to increase the max lease duration, there are a number of other possible workarounds: reduce the publication period, increase the time distributed interval, increase the project publication frequency (reduces amount of change being published each time), decrease the threshold for number of changes needed to trigger publication, or remove baseline publication.

If the error is occurring from Project Summarization, the following other solutions are possible: reduce the number of WBS levels being summarized for the project in question, or increase the size of the summarization period from week to month.

Special considerations: Project Summarizer

The Summarizer job service is unique in that it will automatically create internal "child" project summary jobs whenever an EPS is being summarized. This is because calculating the summary information for the EPS *depends* on the project summarization of the child projects. Because the real work of the summarization is at the project level, this allows the internal child-project summary jobs to be summarized concurrently by any number of summarizer threads, across multiple machines. This allows the summarization of an EPS to be distributed and scale nicely when there are multiple job services machines deployed.

Special considerations: Project Scheduler, Apply Actuals, Leveler

Creating one job at EPS level vs. creating individual jobs for each project

The Project Scheduler, Apply Actuals, and Leveler job services are similar in that performing these actions at an EPS level will cause the service to operate on the sum of the individual activities and resource assignments as a whole. It will **not** create child-project jobs the way the Project Summarizer does. Therefore, any p6services process (thread) that starts working on the job will completely process the job, it cannot be broken up and distributed across multiple threads or machines. This holds true for both manually executed jobs as well as jobs created in Project Scheduled Services. The reason for this is that there may be inter-project dependencies that must be considered in the context of the entire project set in order for the calculations to be correct.
Therefore, if it is desired to distribute scheduling, leveling, or apply actuals processing for an EPS level job, you should instead create individual jobs for each child project that doesn't have interdependencies with other projects. To accomplish this you may need to do some analysis on your activity networks to determine which projects must be scheduled within the same job.

Appendix 1 – Database table JOBSVC and column descriptions

The important columns of the JOBSVC table are described below:

- **job_id** – the unique id for this job record
- **job_name** – Name of the job given by user who scheduled the job, or a generated name if job was generated by the system
- **recur_type** – Indicates whether job is scheduled or should run immediately. It will have one of the following string values which have the following meanings:
  - RT_WebEnabled – Job is a recurring job (scheduled) and is enabled
  - RT_RecurDisabled – Job is a recurring job (scheduled), but is currently disabled

- RT_WebASAP – Job is a one-time job, to be run as soon as possible. Check status_code to see if it has already been run or is pending.

- **job_type** – Job type – Indicates the type of job. The following values have the following meanings:
  - JT_Sched - Schedule
  - JT_Sum - Summarize
  - JT_ApplyActuals - Apply Actuals
  - JT_XERExport - XER Export
  - JT_AsyncRequest - Async Request
  - JT_CollabSync - Collaboration Synchronization
  - JT_InitiationSync - Initiation Synchronization
  - JT_Batch - Batch Report
  - JT_Report - Generate PM Report
  - JT_StorePerPerf - Store Period Performance
  - JT_SyncActThisPer - Sync Actual This Period
  - JT_SyncRemToPlan - Sync remaining and planned values for non started activities.
  - JT_SyncActDuration - Sync actual units and costs to match duration percent complete for activities.
  - JT_RecalcCost - Recalculate the assignment cost for the project
  - JT_ETL - Extract Transfer Load
  - JT_ImportProject - Import Project
  - JT_ExportProject - Export Project
  - JT_SendToFusion - Send to fusion
  - JT_Level - Level
  - JT_ScheduleCheck - Schedule Check
  - JT_OverallocCheck - Overallocation Check
  - JT_StatusUpdateCheck - Status Update Check
  - JT_EnterpriseData - Enterprise Data Service
  - JT_CopyService - Copy Project Service
  - JT_CreateBaseline - Create Baseline Service

- JT_EnterpriseSum - Enterprise Summary Service
- JT_ResourceMgmt - Resource Management Service
- JT_Project - Publish Projects
- JT_Security - Security Service
- JT_ProjectArbiter - Project Arbiter
- JT_Unifier - Send To Unifier

job_data – blob field containing detailed information about how to run the job

parent_job_id – job id of the parent job if this job was automatically created as a result of running another job. For example, summarizer jobs for EPS or portfolios will create child jobs for summarizing each project.

status_code – Status of the job. Values: [JS_Pending, JS_Running, JS_Complete, JS_Failed, JS_Cancelled, JS_Delegated, JS_CompError]

The JS_Delegated status indicates this job has created child jobs and is waiting for them to be completed

worker_host – This field will contain the network host name (and possibly java processid) of the machine/process that was last processing the job. This field will be in the form "hostname::processid" where [::processid] is optional based on version of p6services you are running and ability to retrieve this information.

last_run_date – The date/time that this job was last *completed* (or failed).

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com

Oracle is committed to developing practices and products that help protect the environment

Hardware and Software, Engineered to Work Together