JobScheduler for Oracle

“7x24 solutions for the Oracle enterprise”

The Business Problem

Evolution of Oracle enterprises

From its roots as a decision-support database tool, Oracle has evolved to be the relational database of choice for everything from data warehouses to mission critical systems. Oracle may now be found in use underlying systems as diverse as those in Table 1:

Airline reservations	Sales force automation
Decision support	Customer service
Data warehouses	Call center operations
Accounting	Customer databases
ERP	Web site content management
Medical records	Electronic commerce
Data mining	

Table 1: Uses of Oracle

Originally, these systems were developed in-house, but more recently third-party vendors have modified their applications to use Oracle as their standard repository. Perhaps the clearest sign of this is ERP vendors like SAP and Peoplesoft competing with Oracle's own applications product, all running against the Oracle database.

Need for "batch" jobs

In redeveloping their applications for client-server environments, some vendors took the high road by rearchitecting for a real-time environment. Typically however, the reality of a capacity constrained hardware and network environment has forced them to use some sort of batch or non real-time methodology.

Other vendors merely ported their mainframe applications to a client-server environment and assumed the presence of some sort of batch or job scheduling mechanism similar to that available on mainframes.

Most organization recognize that maximizing ROI on expensive hardware requires off-hours utilization for at least house-keeping tasks, and perhaps more such as long-running reports. In a similar fashion (but perhaps without as many hard analysis) it is clear that automating manual tasks performed by programmers, operations staff, or financial analysts is at least as important as better hardware utilization, and probably leads to greater productivity and job satisfaction. Typically this has led to the recognition of the "batch window" during which batch jobs can or should be run, and simultaneously a realization of the trend towards shrinking this batch window.

As the number of packaged or third-party applications has increased, so has the need for cross-application tools which make sure processes or jobs from disparate systems work together (for example, by ensuring that the General Ledger Posting does not run before all relevant AP transactions have been processed for the day.

Job Scheduling Requirements

Although the title may be different in different organizations, it typically falls to an operations or production control manager to reconcile the conflicting job requirements presented above. In many newer organizations this role may fall by default to the systems administrator, or even the DBA. Initial attempts to handle this may use manual scheduling, use of Unix **cron**, or "vanilla" schedulers included in 3rd party packages (which typically work well only for jobs from those packages).

With some experience, the operations manager will recognize the following as basic requirements for any job scheduling system. Not surprisingly, many of these mirror the needs of traditional mainframe schedulers.

Basic requirements

Oracle Repository-based

This is a fundamental requirement for any Oracle-based enterprise. DBAs or system administrators want to be able to monitor, analyze, and report on job activity. Although built-in reports may provide some of this functionality, standard Oracle tools can tie together the job scheduling component with other in-house developed tools.

Ability to automate jobs from multiple applications

Job schedulers must be able to run jobs from multiple applications and environments, without requiring any change to the program being run. Changes are not only undesirable but also may be impossible in the case of a 3rd-party application. This in turn implies the requirement to:

- pass an unlimited parameters of arbitrary type
- read input from a file in place of terminal or command-line input
- transparently run script-type files (such as SQL or PL/SQL scripts) through a pre-processor

Need to run jobs or groups of jobs in a specified order

Efficient use of a shrinking "batch window" demands that jobs be run in parallel wherever possible. At the same time, some jobs require the output or results of previous jobs and so must be run sequentially. Taken together, these needs imply the ability to run job

"streams" (also called sets, batches or schedules) where some jobs are parallel and others are sequential as illustrated in Figure 1.

Calendaring or "time-and-date" scheduling

Jobs and job streams typically execute on a calendar which mirrors the business calendar. For example, the monthly closing of the books entails running postings, reconciliations, and reports, and finally the close itself. Calendar specifications allow building basic "every Monday" type calendars, as well as more complicated custom calendars. Calendars also need to include running jobs at multiple times or intervals on calendar dates.

Ad hoc scheduling

The power of a job scheduler, and the control it brings to resource utilization can also be extended to ad hoc scheduling. In this environment, resource intensive jobs are "scheduled" to run immediately, thus providing a compromise between real-time needs and not overloading the system.

Job constraints and error handling

Obviously job streams do not always run successfully, either because of problems internal to the job stream (such as a misconfiguration, bad parameters, or an executable program having been misnamed or moved), or because of external problems (a required input file not being available, or a previous job having nor been run). Job stream capabilities require extensive constraint handling (also called dependencies or events). For example, a job to load external AP records into the AP system should check that the external data file is available before starting to run. Optionally, the job should fail if the file is not available within a preset time limit. Other minimal constraints are based on the success or failure of a previous job step, return codes, and time-or-date.

Advanced requirements

The Basic Requirements provide the minimal configuration to run jobs and replace any manual job execution or scheduling system. However, to truly gain the benefit of a job scheduling system and the accompanying automation, there is an additional level of advanced requirements.

Fault tolerance and restart/recovery

Complicated schedules can quickly be rendered useless if a critical component, system, network, or database is unavailable during execution time. Fault-tolerant capabilities enable processing to continue at some minimal level during such interruptions, and for full service to be restored as soon as possible.

The interruption of running jobs is only the most immediate problem from a system crash. The flow on effects of schedule delays can lead to the night schedule ending without critical jobs having been run. Consequently, recovery/restart logic tells the scheduler which jobs are critical and which can safely be deferred.

Dynamic load balancing

If the maximum benefits of a scheduler are to be had by combining both calendar-scheduled and ad hoc scheduling, there arises a danger of load spikes caused by resource-hungry jobs being run during the day (or by users starting long-running eal-time jobs running before leaving in the evening). Dynamic load balancing is essential to allow system administrators to control the number and types of jobs running depending on system load, to minimize disk bottlenecks and CPU "thrashing".

Scheduling across multiple systems

Many open systems environments use multiple identical servers "clustered" together (in a formal or informal fashion), often with shared disk storage. Not only does this provide for some level of redundancy, but it also provides added parallelism when needed. An advanced scheduler should allow such clusters to be treated as a single server for scheduling purposes, so that jobs will run on the first available server, or the server with the lowest load.

Sophisticated multi-level security

Although a job scheduler is a useful tool for just the operations manager, system administrator, or DBA to use, it becomes even more powerful if jobs do not require constant oversight. Much of the promise of open systems is the "opening of the glass-house" and the exposure of capabilities to end users. A sophisticated job scheduler should provide user-friendly features for defining and running jobs, all with appropriate security. For example, accounting managers might be permitted to define accounting jobs, accounting users might be able to submit them, but only operations personnel could control the scheduler itself, including when and how those jobs run.

Parameter validation and automation

Effective use of a job scheduler by end-users requires user-friendly features such as context-sensitive parameter validation and lookup lists. For example, an accounting user in the ABC subsidiary might only be allowed to run reports for divisions within that subsidiary, and so should be presented only with the relevant list of divisions when submitting reports.

Output Management

Scheduling and running jobs without efficient output management is self-defeating. Jobs can produce reams of output, and quickly fill up available disk space. Schedulers must have sophisticated output management which:

- Allows online viewing of output. This ensures that users and operators can preview reports to ensure they are correct before printing or distributing them
- Provides secure access to printers. An obvious example is that only certain users running certain jobs should be allowed access to the check printer.

- Monitors printer status and can redirect jobs if a printer is down. You should be able to define alternative printers (of the same type) that can handle a print job if the designated printer is down.
- Manages disk space. Output files from old jobs must be aggressively purged or archived according to criteria like age, size, print status and so on.

JobScheduler for Oracle capabilities

JobScheduler for Oracle was designed to provide both industrial-strength capabilities previously found in mainframe schedulers, and also to leverage the benefits of open systems architectures.

Basic requirements

Oracle Repository-based

JobScheduler for Oracle uses the Oracle database for storage of all objects, as well as communication between Master and Agent Schedulers, ensuring a persistent store for all job-related activities. JobScheduler provides standard Oracle reports for all important scheduler objects (such as jobs, jobs streams, programs, and so on), as well as dynamic queries such as failed jobs of certain types. However, because it is Oracle repository based, DBAs can run their own ad hoc queries to interrogate the job database in their own way.

Ability to automate jobs from multiple applications

JobScheduler for Oracle can run any job without change by simply registering the executable program file and any number of parameters.

JobScheduler's powerful **program types** enable you to predefine preprocessors and parameters for commonly run jobs. For example, you can define SQL*Plus or PL/SQL scripts as jobs using JobScheduler's predefined SQL*Plus program type which automatically logs into SQL*Plus and opens a spool file..

"Interactive" jobs can provide a special challenge, especially those that read input from the command line. But JobScheduler's Autofile Manager allows you to define an input file that provides input commands to a job to replace the command line.

Need to run jobs or groups of jobs in a specified order

JobScheduler's Job Streams give you complete control over the ordering and execution of your jobs. Simple sequential job streams can be defined with one checkbox, and more complex parallel or networked job streams can be defined with a combination of sequential execution and job constraints. You can even build your job stream in a modular fashion (allowing you to share job stream definitions) by nesting job streams to an unlimited depth.

Calendaring or "time-and-date" scheduling

JobScheduler for Oracle includes extensive calendaring. Not only can you share calendars among different jobs and Job Streams, but calendars can be as simple as "Every Monday" or as complicated as you like. And using Calendar Periods you can define jobs to run at varying intervals, between arbitrary times on any date for the ultimate in sophistication.

Ad hoc scheduling

JobScheduler allows you to treat ad hoc jobs with any priority you define relative to regularly scheduled jobs. You can define separate queues for ad hoc jobs to limit the number or load, or time of day when they can be scheduled. And regularly scheduled jobs can be changed to ad hoc jobs with a click of a button if they need to be run immediately.

Job constraints and error handling

JobScheduler's job constraints are the core of running jobs and job streams. JobScheduler provides six different types of constraint for fine-grained control of jobs:

- Job status: execution of following jobs can be based on the completion, success, or failure of a previous job
- Job return code: provides greater control than status, with different job paths depending on return code from a job
- Alarms: jobs will run if a previous job has raised an alarm because of its completion status (allowing you to define generic error handlers for different types of job failure)
- SQL conditions: let you specify that a job should run if a SQL condition is true or false (for example, if needed data has been loaded into an interface table)
- Files: jobs can wait for a file to be transferred from another system (for example, and interface file which then will be loaded into the database) or a file to be deleted (for example, a lock file which indicates a backup is in progress)
- Calendars: for situations where the same job stream should run with minor variations on a few days, you can define a single job stream and then include a calendar constraint to run or not run a sub-job based on the particular day. For example, the month-end close might run slightly differently at year end.

Furthermore, job constraints can be combined with if-and-or logic for complete flexibility in job execution.

Advanced requirements

Fault tolerance and restart/recovery

JobScheduler's advanced fault-tolerant features include:

- Master Auto-restart
- Database tolerance
- Hibernate mode

Auto-restart exploits JobScheduler's advanced processor-independent client-server architecture. In normal operation, the Master Scheduler provides a single point-of-control for scheduling decisions, and then sends jobs to the appropriate Agent Scheduler for execution on a local or remote system. If the central system should fail, thereby crashing

the Master Scheduler, all Agent Schedulers immediately recognize the problem. You identify candidate Agent Schedulers, one of which "clones itself" and restarts the Master Scheduler, with accompanying clean-up and recovery/restart logic for jobs that were running at the time of failure. And the Master Scheduler even re-synchronizes itself with whatever jobs are currently running on the Agents!

Database tolerance: Network dropouts are a fact of life, especially in widely-distributed operations. But JobScheduler for Oracle allows for temporary interruptions in service. Whether it's a hardware failure or a SQL*Net problem, if JobScheduler for Oracle is unable to access the Master Scheduler database, it continues normal operation (including job execution, monitoring, and completion) while retrying the database at increasing intervals. When normal service is restored, the Scheduler synchronizes the database with its stored status.

Agent Scheduler Hibernate: If the database or database server should fail, you don't want to lose all your job execution information. With Database Tolerance, JobScheduler for Oracle attempts reconnections at increasing intervals. You can specify a retry limit after which JobScheduler for Oracle assumes the interruption is not a temporary network interruption, but rather a more serious database problem. JobScheduler for Oracle writes all its schedule information to a "hibernate file" and then shuts down gracefully. When the database or network problem has been fixed, you can restart JobScheduler for Oracle which rereads the hibernate file and resynchronizes the Master Scheduler database with it. If you elect not to restart an Agent Scheduler on that system, you can still run a synchronize utility to post the hibernate file to the database.

Dynamic load balancing

JobScheduler's default behavior is to use static load balancing, which provides the simplest way to get started running jobs. But if load spikes are a problem, or if you are consistently maxing out your system, you can turn on dynamic load balancing. Use any load variable you want; something as simple as the **uptime** load average, or more complicated statistics from third-party tools like PATROL or Compuware. Define a job to load profile in your queue definitions which "throttles down" queues as the load increases, allowing you to reach a steady state and maximize the throughput. Of course, you can still automatically or manually override load limits for running critical jobs.

Scheduling across multiple systems

JobScheduler's powerful queue mechanism allows you to restrict programs to run on a single scheduler (and thus in turn a single system), or to multiple systems in a cluster. And even more importantly, JobScheduler can combine this capability with user restrictions on who can run on which system for maximum security.

JobScheduler's easy-to-read Control Center lets you see at a glance what the scheduler and queue load is, and what is running where.

Sophisticated multi-level security

JobScheduler uses multiple security layers to ensure that you can control who can run what jobs.

Oracle-user security restricts access to the Scheduler to acceptable Oracle users. You can use the JobScheduler user profile to control how many jobs users can run simultaneously, and the security groups to restrict users' access to running specific jobs, or using certain printers.

Menu-based security ensures that users only see the forms and screens that you specify; the pre-programmed Admin, Super-User and User roles package the most common features together.

Security levels control what functions users can access in a specific form. Six different security levels, from User to Administrator, provide increasing capabilities, from being able to submit and control only one's own jobs, through to being able to control the whole Scheduler.

Parameter validation and automation

JobScheduler gives you complete control over parameters to jobs, to minimize the instances in which jobs run with incorrect parameters. You can specify:

- Type: (Character, Date, or Numeric)
- Display: Hidden, Read-Only, or Visible. For example, Read-Only parameters are displayed in the job submission, but may not be changed by the user
- Min-Max validation
- Table-Column validation
- Mandatory/Optional
- Default

And JobScheduler allows you to reuse the parameters entered for a previous run (or change selected ones) and resubmit the job.

Of course, automatic submission of jobs on a calendar is vastly preferable to manual submission. But many programs' parameters change depending on submission date or other data. JobScheduler's **Automatic Parameters** allow you to get the best of both worlds! You define Automatic Parameters using SQL statements, which can even include references to other Automatic Parameters.

A simple example is a report which you want to run with a parameters of the first day of the current week. You can define an Automatic Parameter called **First Day of Week** which uses a simple SQL statement to compute and pass the first day of the current week whenever the job is run. You need never submit the job manually again; JobScheduler correctly compute the parameters automatically whenever the job is run.

Output Management

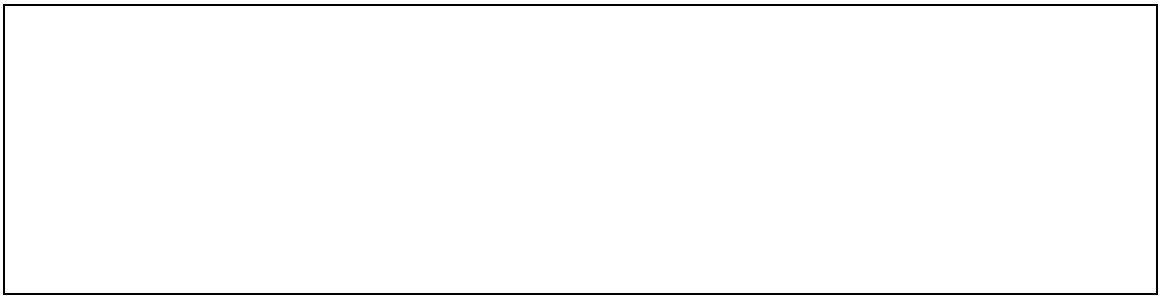
JobScheduler's output management features include:

- Printing instructions which can depend on job status (so that you don't print output from jobs which errored)
- Multiple online viewing techniques for users, ranging from a quick preview to a more extensive "copy to local disk" option (perhaps as a precursor to loading into a spreadsheet)
- Print styles which can be matched with printer types to ensure that the right output goes to the right printer (no more sending HP Laserjet output to a Postscript printer)
- Printer security integrated with security levels (which include lists of users and allowable programs)
- Automatic print redirection if a printer is down
- DiskSentry output management and purging based on multiple criteria including job status, print status, size, age, or manual selection

JobScheduler's output management features ensure that you can always find the space to run necessary jobs!

Product Architecture

As Figure 2 illustrates, JobScheduler for Oracle is architected in a portable, scalable fashion.



Oracle repository

All job objects and information is stored in an Oracle repository which is constantly updated to provide up-to-the-second status in the event of a system crash or interruption. The Scheduler accesses the Oracle database intelligently and with minimal overhead, reading and writing only for changes to job information.

Master and "light-weight" Agent

Each scheduling group consists of one Master Scheduler which maintains the schedules, queue information, and job streams, and also runs jobs locally on its system. Jobs are executed on other systems by light-weight "Agent" Schedulers which report job status back to the Master Scheduler through a message queue implemented in the Oracle database.

In the event of a schedule interruption such as a crash of the system running the Master Scheduler, one of the Agent Schedulers can automatically "clone" itself as a Master Scheduler to continue scheduling jobs with minimal interruption. And if Agent Schedulers should shutdown, jobs can automatically be rerouted to alternative systems.

Schedulers written in "C"

Master and Agent Schedulers are written in "C" for maximum portability across any Unix platform (99.9% of the code is unchanged during porting) or Windows NT, using Oracle-standard Pro*C calls and stored procedures for database access.

Developer/2000 interface for consistency

JobScheduler's Windows 95 interface is written in Developer/2000 for consistency and ease of use with your internally developed Oracle systems. (A free runtime version of Developer/2000 is available from Oracle for those who do not have the developer copy).

Conclusion

JobScheduler for Oracle provides industrial-strength job scheduling for Oracle enterprises needing 7x24 solutions. Not only does it provide the basic job scheduling capabilities that enterprises expect from their mainframe legacy, JobScheduler also delivers advanced functions that significant improve throughput, performance, and ease of use.