Open Source Job Scheduler

JOB SCHEDULING À LA CARTE

Planning and scheduling jobs can mean a lot of work, especially if they are spread across multiple machines. Here's a tool to make that task a lot easier.

BY JAMES MOHR

Tinka, Fotolia

The ability to perform a certain task at a specific time or at regular intervals is a necessary task for sys admins. The original cron daemon offers an easy method for job scheduling on Unix-based systems. Although cron has seen a number of improvements over the years, even the newer versions are designed for very basic scheduling. An administrator who wants to do anything unusual must either create a wrapper script or build the additional functionality into whatever script is started by cron.

Imagine how much time you could save if you no longer needed to create wrappers, hack your scripts, or do anything else to get programs to react to error conditions and run at exactly the time you need – in exactly the order they should. Several commercial products offer this functionality, but they can take a big bite out of your IT budget. Luckily, the open source world also provides solutions for beyond-cron scheduling.

In this article, I will explain how to get started with a powerful alternative: the Open Source Job Scheduler.With any scheduling software, the primary administrative unit is the *job*, which is typically a script or program started by the scheduling software. In many cases, cron is sufficient to handle the most simplistic scheduling requirements, such as running a certain job once a day (i.e., backups). Even jobs that need to run at more frequent intervals (every 15 minutes), less frequently (once a month), or even on specific dates (the first of the month) can be handled by cron.

When you start dealing with dependencies of any kind, you quickly begin to see the limitations of cron – for example, if you want to start a specific program after a certain event occurs. In my work, I have a number of job *chains* that consist of up to a dozen individual jobs that must be run in a precise order, and each job can only run if the previous job in the chain was successful. If one step fails, the execution of subsequent jobs would create major problems. Even the newest versions of cron cannot handle this, especially if you need to be able jump into the middle of the chain occasionally and start a job from there.

Events that start jobs can be more than just specific times or the completion of other jobs. Often jobs simply wait until either a specific file is delivered or anything at all arrives in a predefined directory. Naturally, on many occasions, specific jobs must be started on demand rather than waiting for a specific event.

Many tasks need to be managed on multiple machines, so the better scheduling software allows you to manage all of your machines from a central point, remotely start jobs, and so forth. To distribute crontabs to these remote machines, you could configure *rsync* or use some other mechanism, but this quickly becomes an administrative nightmare when the configuration is different across several machines, when jobs need to be started manually often, and when other tasks exist that are not part of cron's functionality.

Rabbit to the Rescue

Although you can install open source versions of cron on Windows machines, dealing with different operating systems causes even more problems. For example, some Unix dialects do not support a central */etc/crontab* file, so you need to set up files for individual users. Cron

was a useful tool in its time, and often still is, but the requirements of many companies rule it out.

The limitations of cron have not gone unnoticed, and a number of products have come on the market. Commercial products can cost a small fortune and are often licensed on the basis of the number of servers or, in some cases, the number of scripts you start.

Problems with job scheduling and commercial software licensing have not gone unnoticed by the open source community. One amazing solution is the Open Source Job Scheduler, developed by Software- und Organisations-Service (SOS) GmbH in Berlin, Germany. Versions are available for Linux, Solaris, HP-UX (PA-RISC, IA64), AIX, and Windows. It also supports several different databases, including DB2, Oracle, MS SQL Server, PostgresSQL, and MySQL.

Depending on your needs, two different licenses are available: GPL and Guaranteed License. Both license types provide the same basics, including all of the functionality, source code, and upgrades. Note that the product is the same with both licenses.

Although some products provide more features in the commercial version, SOS Managing Director Andreas Püschel sees their business focused on support and service and not the product itself. Also, SOS does not market their product – or even their services – in the traditional sense. When demonstrating the product, the goal is not to convince potential customers, but rather to show what the product can do and let the customers decide for themselves whether the product fills their requirements. Simply seeing what this product can do will convince a lot of people.

The commercial version also provides a "responsible person" for each service call with guaranteed response times, and feature requests are given a higher priority for possible implementation. Also, this version does not have the restrictions of the GPL, so you can bundle it with your own application, for example, without having to adhere to the GPL.

Unlike most open source and commercial software, SOS also provides a limited two-year warranty, as well as an indemnity agreement. According to Püschel, the company feels obligated to give the customer what they pay for, and this also extends to discrepancies between the product and documentation.

The central component of the package is the Job Scheduler engine, which runs on every machine on which you want to *schedule* jobs. This method is different from executing the job, and the documentation provides a couple of ways to provide for remote execution, one of which is to have a scheduler installed on the remote machine acting as a slave for the first machine. However, if necessary, the other servers can function as a full scheduler, as well. This mechanism can be expanded to enable load balancing across multiple servers. A job chain is managed from one machine and distributes the requests to the other machines.

One key aspect to consider is the so-called *order*, which is a token or flag that is passed between jobs. Depending on their configuration, jobs cannot start until they have been given their order.

In the simplest form, orders are like a baton in a relay race, handed from one job to the next. However, they can also contain parameters that are passed between jobs (e.g., the current file being processed). Also, you can configure orders with specific start times so they are automatically generated by the system at the specified time, and then the respective job chain can start.

Note that to be able to react to an order, a job must be configured to accept them, but the order can only be associated with a job *chain*, as opposed to an individual job. That is, the order is passed from job to job within the job chain, but it is associated with the chain and causes the job chain to start. If you want a single job to start at a specific time, for example, this can be done within the job itself. Another key component is Hot Folders, which are directories that the scheduler monitors for changes, such as new or modified jobs.

Jobs can be configured from the Job Scheduler Editor GUI (Figure 1; hereafter called the Job Editor), or XML files can be edited directly. The Job Editor GUI is a Java-based application that you can use to configure the various jobs, chains, and other aspects of the system. All of the server configuration information is stored in XML files, and all you need to do is open up the respective XML file in the Job Editor to make your changes. From my experience with other job schedulers, being able to use *vi* on configuration files is more than a blessing when having to do massive changes.

The XML files can be copied to remote machines and saved via FTP directly from the Job Editor. When they land in a Hot Folder, the files are immediately available to the scheduler engine on the remote machine.

The Job Editor GUI is not as intuitive as it could be, and the purpose of many of the fields is unclear at first. Because the documentation leaves something to be desired, explanations for many of these fields was simply not to be found. In some cases, I could still figure out what was meant by the descriptions of the XML files in the documentation.

Although the configuration information is stored in XML files by default, you can configure the scheduler to use a number of different databases. These jobs are called "managed jobs," and every scheduler you set up can be configured to access the jobs in the database, so you do not need to copy the files manually.

The day-to-day operation is handled by the Job Scheduler operations GUI (hereafter called the Operations GUI), which is run through a web browser, thus allowing you to manage your jobs from almost any machine. With this GUI, you can monitor not only jobs, but also start, stop, handle errors, and many other functions.

The Job Scheduler also provides an API that allows you to manage and control jobs externally. The API supports several languages, including Perl, VBScript, JavaScript, and Java. Surprisingly, PHP is not supported, despite the ability to manage jobs from a web browser and the documentation of sample PHP scripts.

Scheduling the Installation

For the examples in this article, I used version 1.3.4 for Linux, which you can download from the Job Scheduler site at SourceForge [1]. If you plan to use a MySQL database, as I did, note that the Job Scheduler does not provide a JDBC driver for MySQL, although one is provided for Oracle and other databases. The MySQL JDBC Driver can be downloaded directly from the MySQL website [2]. Simply input the path to the appropriate *.jar* during the installation.

Before you start, I recommend that you read the PDF installation guide that is included in the package. Also, several other PDFs are on the company's website [3] that go into more detail about various topics. Be warned: To get even the most basic information from the documentation, you need to be somewhat familiar with object-oriented programming and XML because the documentation provides almost no background information in these areas. Also, the documentation is not well organized, so expect to do a lot of searching. Although the documentation is extensive, it's not easy to use, which is something the company plans to improve.

On Linux, installation is through a Java installer and, if done as a normal user, the default is to install it in *$HOME/scheduler*. If you install it as root, it ends up in */usr/local/scheduler*. After you install the product, you will find a README file that recommends you don't install the Job Scheduler as root, but nothing in the Job Scheduler Installation and Configuration guide mentioned this. To avoid potential problems, I re-installed as a normal user.

During the install, you are prompted for the database type and connection information. Whether "database parameters" are the connection parameters once the database is running or connection parameters to create the database is unclear. Unfortunately, it's recommended that you create a database and user for the Job Scheduler to use, but this is first mentioned *after* all of the installation steps. Creating the database by hand and assigning DB privileges before you start the installation does the trick.

The installation is fairly intuitive, but it takes a few minutes to create the database tables and complete. If you are planning to install the scheduler on multiple machines with the same parameters, you are prompted to create an automated installation script after the end of the first installation.

Regardless of whether you install the Job Scheduler as root or as a normal user, you will need to start the scheduler by hand. Control of the Job Scheduler is done like a typical rc script:

```
$HOME/scheduler/bin/jobscheduler.sh start
$HOME/scheduler/bin/jobscheduler.sh stop
```

If you want to start the scheduler automatically when the system boots, I suggest that you create a specific user just for the Job Scheduler and then create an rc script that does an *su* to that user and starts the scheduler. Should jobs need to be run as root or another user with more privileges, you can set up an appropriate *sudo* environment.

Creating Jobs

To create jobs, you can either edit the XML files directly or through the Job Editor GUI. On Linux, run the *jobeditor.sh* script, which is located by default in */usr/local/scheduler/bin* or *$HOME/scheduler/bin*.

As an example, consider the typical task of creating a backup of your system configuration. For now, I'll assume that you want to do this daily with the script */usr/local/bin/config_backup.sh*.

First, start the Job Editor and select *New* | *Hot Folder Element* | *Job* (Figure 1). On the first form, begin inputting the basic information for your job. For this example, simply input the *Job Name* "Configuration Backup"; in the *Job Title* field, add a description or leave it blank.

In the left-hand panel, click *Execute* to input the details about the program or script you want to start. Because you want to execute an external script, select the radio button *Run executable* and input the complete path to the script named above. Here, you also can define additional parameters that are passed to the script. For example, if the backup is to compress the files it backs up, you might add a *-c* here.

Also, you could include the individual execution steps by selecting the *Script* radio button and the type of program code, then inputting the source code in the appropriate box. Note that this is more than the name of a script and can include programming constructs based on the language you select (Figure 2).

To save the job, press the *Save* button or choose *Save* from the File menu. The

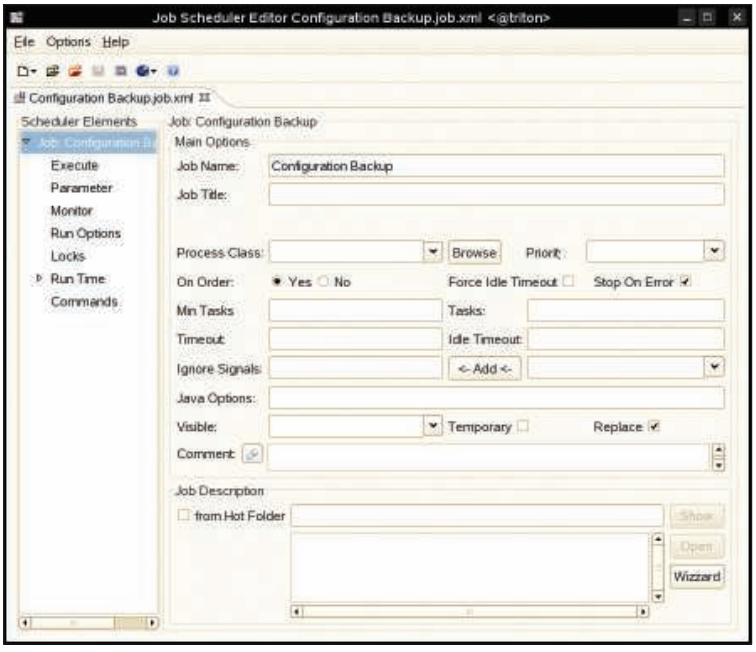

Figure 1: The Job Scheduler Editor GUI.

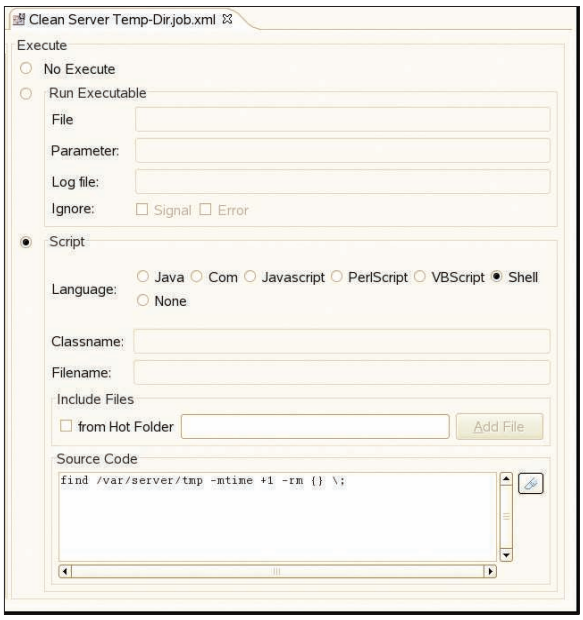

Figure 2: Inputting the source code directly into the the Job Editor.

first time you save the file, you are prompted to input the file name. To do so, navigate to the *../config/live/* directory and save the file as *Configuration Backup*, and the *.xml* extension will be added automatically.

Because you saved the file into the *live* directory, it is immediately visible to the system. This directory is pre-defined as a Hot Folder, which the system reads regularly. At this point, you have not scheduled the job but simply added it to the system. To start the job, you need to run the Operations GUI by pointing your browser to *http://localhost:4444*.

When you connect from the browser, you are presented with a GUI similar to that in Figure 3. To see the details of the job, double-click on your backup job in the left-hand column. To run the job immediately, click on the *Job menu* button and select *Start task now*.

Because you could easily start the script from the command line, this is pretty unspectacular. If you go back into the Job Editor and click the *Run Time* entry in the left panel, you can select a time when this job should run. To do so, click on *Everyday* and then define a new period by clicking the *New Period* button. For *Start Time*, input something like *09:00* in the *Single Start* field. Now click the *Save* button and this new configuration is active – the job will now start every day at 9:00am.

In terms of creating jobs, so far the only thing the Job Scheduler editor provides that cron does not is a nice GUI – but you have only scratched the surface. When you begin working with with job chains, you will start to see the power of job scheduling.

First, assume you have created a second job that does a database backup, which you want to run immediately after the configuration backup has completed. One alternative is to create a single script that first does the configuration backup and then immediately starts the database backup. However, job chains come in handy in many more complex situations that you cannot simply implement with a single shell script, which I discuss later. For simplicity's sake, I'll stick with these first two jobs.

As with the first job, create a new Hot Folder element, but select *Job Chains*. Here, you input the *Chain Name* and, if desired, a *Title* (i.e., a description). Because each element in a chain is referred to as a Node, you need to add a *New Chain Node* next by clicking the respective button. If you know the name of the job, you can input it manually or use the *Browse* button to search for the job.

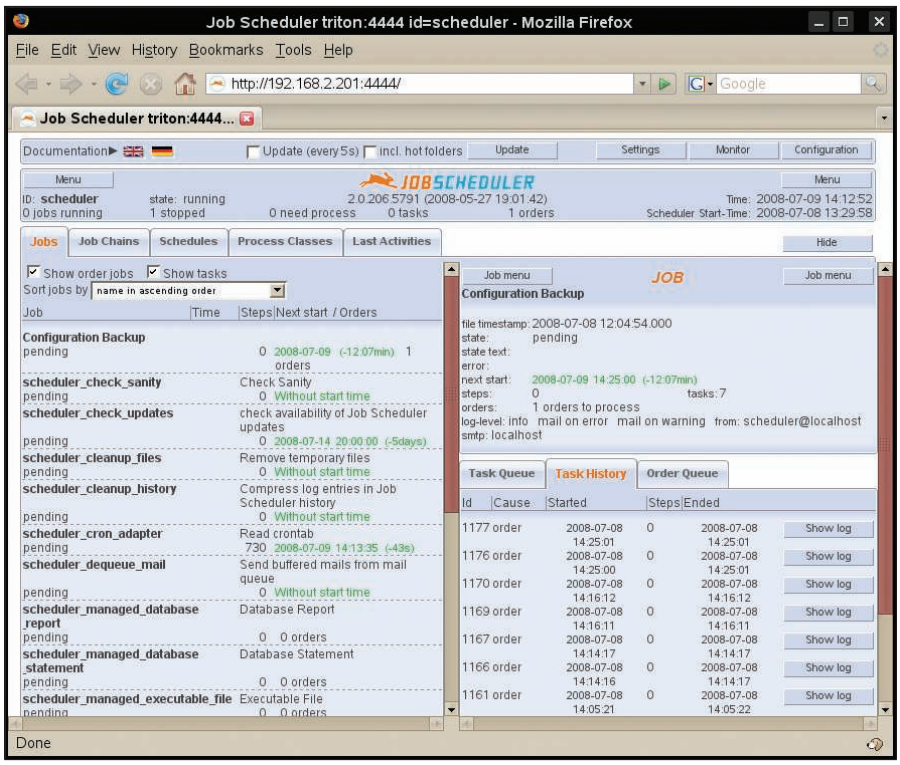

Figure 3: The Job Scheduler operations GUI.

In the *State* field, you can define a state for this step or job node. By defining states, you can define a more complex job flow. For example, you could define a state named "Error," and if one of the steps encounters an error, you immediately jump to that job, skipping all of the other jobs. Also, you could have different error jobs run for each of the various steps.

Although creating a job chain consisting of just a single job has certain advantages, don't stop there. As I mentioned, you want a backup job chain consisting of two steps, so you'll create a second chain node with the database backup job and configure it similarly to the second job. In the example here, you defined the first Node as state *Start* and the second as state *End*, although this is not really necessary.

When you click on the *Save* button, you will see the new chain under the *Job Chains* tab in the Operations GUI. Clicking on the *Show Jobs* checkbox displays the individual jobs in your chain. Double-clicking on the chain opens the details panel on the right-hand side, as it did with the single job.

At this point, the job chain is still not going to be run because it needs its marching orders, which you can create manually by selecting the *Job Menu* button on the right and selecting *Add Order*. A new window pops up and allows you to define various characteristics of the order, such as the order ID, start time, and even the state to which the chain should jump. In this example, just leave everything blank, and the system will create an order ID for you.

If you did not define any conditions, the chain starts immediately; however, you could have defined a *Time Slot* for either of these jobs and the scheduler would wait until that time was reached before starting the job.

Note that the jobs must be able to accept orders to react to them. This step is done in the configuration window for the individual jobs. In the Main Options window is an *On Order* radio button that needs to be set to *Yes*; otherwise, the order will not start the job.

So far, you have started everything manually, more or less. Because you need to define an order in which to start the job chain, you obviously need a way to create orders dynamically. One way would be to create an order at a specific

time, which in turn triggers the job chain. As you might expect, this is done though the menu *New | Hot Folder Element | Order*. After you name the order and, in the Job Chains window, select the specific job chain you want to associate with this order. Note that an order can only be associated with a single chain. Then define a new *Time Period* and a *Single Start* period of *09:00*. To activate the changes immediately, you need to store it in the *config/live/* directory when you save it.

When you return to the Operations GUI, you will see that an order is now associated with the job chain you created. Below the order, a *next start* entry shows the date and time that this order will start. Because it is already after 9:00am in this example, the date is tomorrow. Had you used a different start time, such as the first of the month, the *next start* would be on that date.

Controlling Start Times

In defining the time period for jobs, chains, and orders, you have a couple of choices. First, you can define a specific time slot in which a job can start – for example, a daily account reconciliation after all of the database imports for accounts receivable and accounts payable are completed. However, because of the load this job causes on the system, it should not start until after 11:00pm, even if the other jobs are completed.

The second way is to define a specific start time. For example, the account reconciliation should always start at 11:00pm, or you want the account reconciliation job to run repeatedly every 12 hours. Naturally, if you have a job chain in which the jobs need to be run sequentially, then hard start times of the individual jobs probably won't be needed. However, you can define an order that is started every day at 7:00am, which in turn starts a job chain.

Note that you are not just limited to running jobs at specific hours but can configure a job or order to run on specific days of the week or days of the month, or according to a more complex definition, such as the second Monday of the month, the third to last day of the month, and even days such as January 8 and February 16, but no other dates.

Specific start times for jobs and job chains are obviously a useful feature, but it is not always possible to know in advance when a job needs to be started (e.g., database imports that need to wait until a specific file is delivered). The more direct method is to write the job script in such a way that it exits if the file is not there. However, if your system is then loaded with unnecessary log entries and so forth, you can quickly lose sight of important events. To solve this problem, the Job Scheduler allows you to set up "watch directories." As the name implies, these are directories that are watched for specific files or even based on regular expressions (Figure 4).

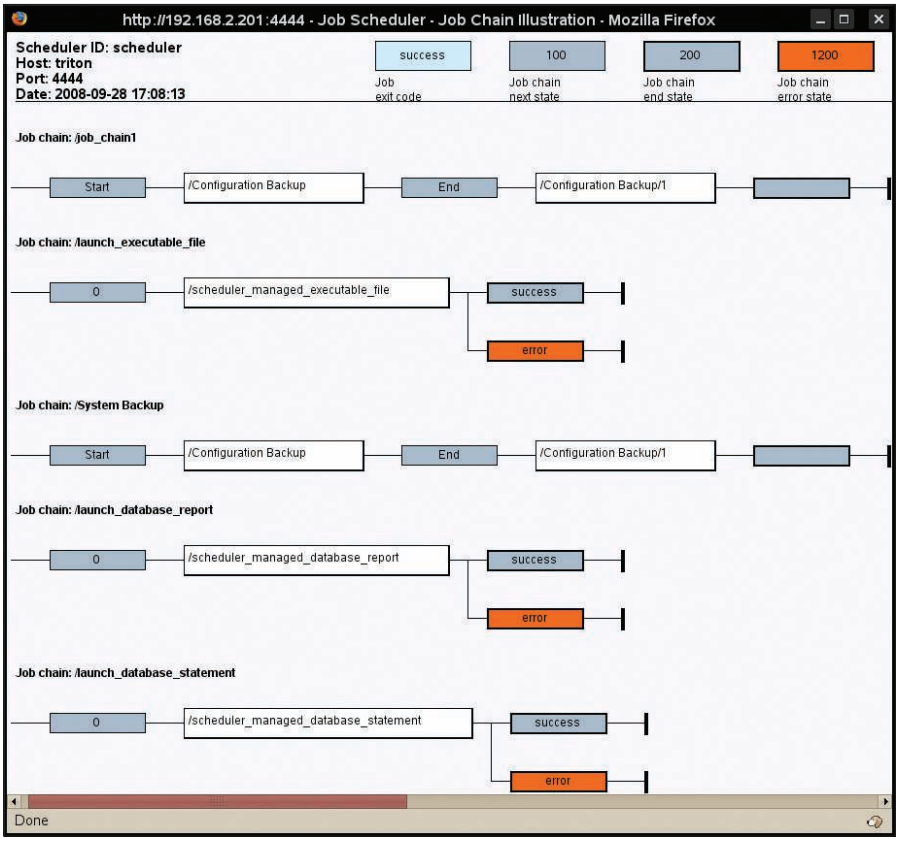

Figure 4: Job Dependencies can be shown using the Job Chain Illustration

Defining Your Own Next Order

After I got past preconceptions and misunderstandings resulting from my experience with other scheduling products, I became more and more fond of the Open Source Job Scheduler. When I got the hang of things, the product was actually easy to configure and administer.

My experience with SOS GmbH itself was extremely pleasant. From the receptionist, through tech support, and up to the managing director, everyone I talked to seemed to be convinced of the quality of the product and the company itself. I became impressed with the company after I identified a bug and the patched Java JAR file was on their server in less than a day!

Even with a small network, the Open Source Job Scheduler provides useful functionality. With larger installations, it is almost an indispensable tool. ■

INFO

[1] Open Source Job Scheduler: *http://jobscheduler.sourceforge.net/*

[2] MySQL JDBC driver: *http://www.mysql.com/products/connector/*

[3] Software- und Organisations-Service GmbH: *http://www.sos-berlin.com/scheduler*

THE AUTHOR

James Mohr is responsible for the monitoring of several datacenters for a business solutions provider in Coburg, Germany. In addition to running the Linux Tutorial web site *http://www.linux-tutorial.info*, James is the author of several books and dozens of articles on a wide range of topics.