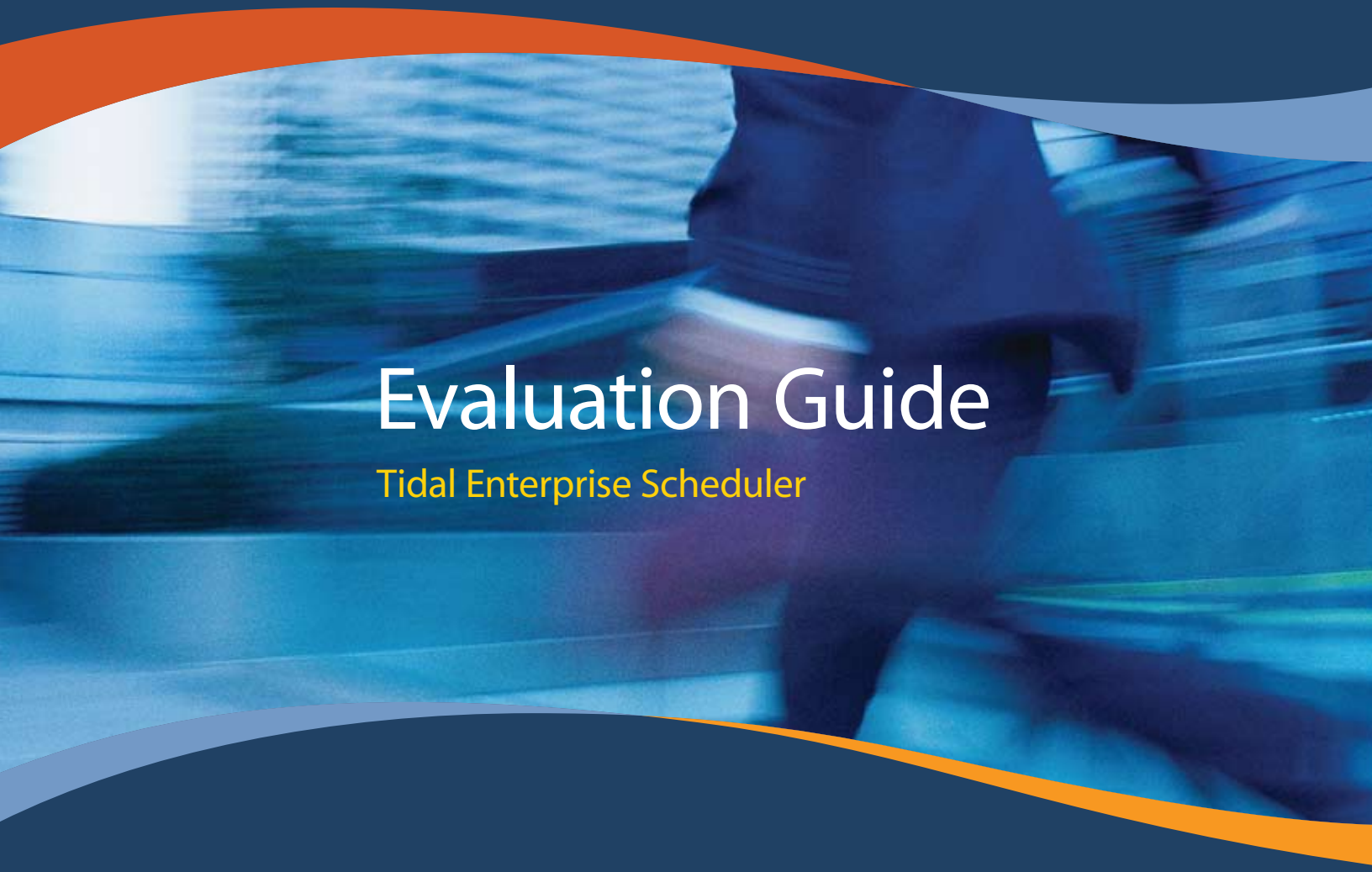


Evaluation Guide

Tidal Enterprise Scheduler

TIDAL® SOFTWARE

Tidal Enterprise Scheduler

Purpose

The purpose of this document is to discuss the requirements for job scheduling software in the modern distributed enterprise. This guide provides a comprehensive list of key job scheduling features and a description of each feature; a companion checklist is also available. The checklist is designed to assist those actively evaluating job scheduling and event automation software by providing a tool for quickly and efficiently gathering information on products under consideration.

To set the proper context for a detailed discussion of key job scheduling features, this document includes some background and history on the evolution of the job scheduling marketplace. This background will help the reader understand how changing business and industry requirements impact the job scheduling arena. Among the handful of disciplines that routinely take place in the data center, job scheduling may be the most important of all. This is a bold statement, given that job scheduling competes with other important systems management functions like file backup, network management and security.

While these are important disciplines in their own right, there is no arguing that, depending on the size of the enterprise, a job scheduler is routinely managing thousands (or, in many cases, tens of thousands) of individual business processes every day. In fact, the number of processes involved is so large that a manual approach is completely unfeasible. Custom job scheduling solutions that rely on native operating system utilities such as CRON, NT Scheduler, PERL and VB scripts quickly become unworkable and collapse under the weight of their own unmanageable 'spaghetti code.' Given this backdrop, it is easy to see how job schedulers are an indispensable part of your IT infrastructure.

Background

The discipline of job scheduling was first established in the 1970s when it became a key strategic infrastructure component for large mainframe-oriented data centers. A variety of products were created and extensively marketed until it became widely accepted that a robust job scheduling tool was required if you were to effectively manage applications on the mainframe. During this period of mainframe-centric computing, a common understanding of the key features of job scheduling began to emerge. When Unix began to make inroads into mainstream data centers in the mid-1990s, IT managers widened their search to job scheduling solutions for managing the distributed environment.

As the shift to UNIX began, few of the existing mainframe vendors created new job scheduling offerings to fill the void. Instead, the mainframe vendors and many mainframe-oriented data centers experimented with attempts to manage distributed workload from the mainframe. As UNIX continued to make inroads into the data center, a new group of competitors entered the market with products created expressly for managing distributed job scheduling.

As the two competing approaches were deployed, it quickly became apparent that the products create expressly for the distributed environment were a far better approach, and the mainframe approach to managing distributed workload was relegated to a small market segment. Even so, many mainframe data centers still cling to the elusive dream of managing all workload regardless of platform from a single console. Unfortunately, few if any have been able to achieve this goal and few vendors appear focused on the issue.

The early distributed offerings proved to be reasonably robust and did a passable job of mimicking mainframe scheduling features and functions, but they suffered from being first generation products. Eventually, all of the prominent vendors of these products 'hit the wall' in terms of scalability, flexibility and ease of use and were ultimately acquired in the late 1990s by mainframe companies who were still grappling with the unique issues and challenges of the distributed marketplace.

During the period that the first generation products were being acquired, newer competitors began crafting more advanced solutions for job scheduling in the distributed environment. These newer vendors had several distinct advantages over their first generation counterparts including:

1. **Improved development technologies.** The early products were generally challenged because of the limitations of the base technology used to develop them. By the early to mid-90s, development tools for the distributed environment had not really matured sufficiently to be of significant value,

so products of that genre were generally built on older, harder to maintain code bases and used the UNIX Motif standard for their graphical user interface. In fact, a number of the first generation products were actually ported from other proprietary platforms in the early 1990s, with the usual complications caused by porting aging software products.

The combination of the older, inflexible code bases, lack of robust development tools and the lack of any GUI standards created problems as the market began to mature and the requirements for distributed scheduling began to change. These first generation products could not keep pace with these changing requirements and began to falter in terms of scalability, ease of use, ease of installation and configuration, and maintainability.

2. **An understanding of cross-platform development.** Like their mainframe predecessors, the early UNIX products essentially had a single-platform orientation: the only difference being that the platform was UNIX instead of MVS or OS/390. This problem came into focus when Windows 95/98 took over the desktop and Windows NT/2000 began to invade the data center.

   While the early UNIX products had the initial lead on Windows NT/2000, their shortcomings for true cross-platform, distributed scheduling became evident as more and more companies attempted to incorporate Windows NT/2000 into their data center operations. Newer generation of job scheduling products are capable of exploiting the ease of use of the Windows desktop and newer development technologies. These newer products also reflect a deeper understanding of the rigorous requirements for building a true cross-platform solution that incorporates UNIX, Windows and other platforms.

3. **A more mature marketplace.** The latest generation of products has been developed with large distributed workloads in the form of applications like SAP R/3, PeopleSoft and Oracle already running in production. The previous generation of products was built on older technology and without the benefit of being battle-tested with significant workloads. Consequently, these early scheduling tools did not have the power and scalability to manage large distributed workloads when it inevitably became a requirement. The shortcomings of these products may not have been readily apparent for several years, at which point their users were then forced to look for a replacement.

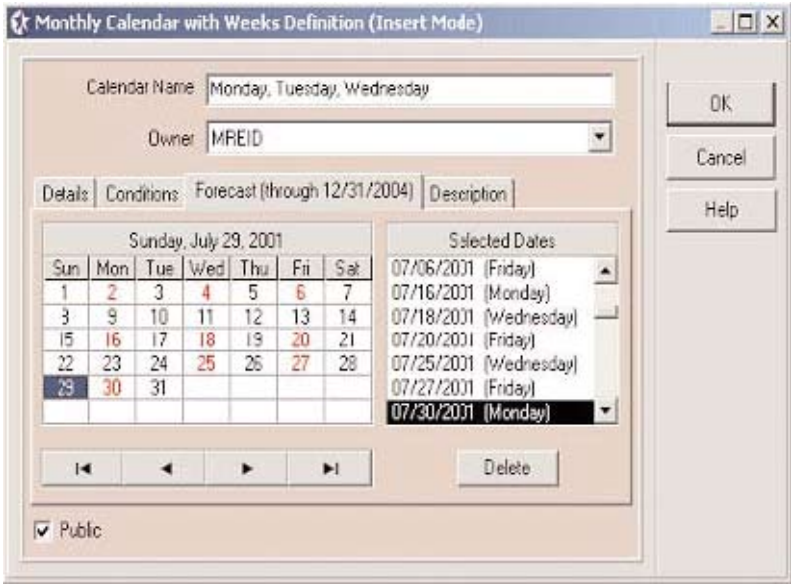
   Many companies are in this position today – looking to replace their first generation job schedulers because of scalability and maintenance issues. The latest generation of products is designed to manage tens of thousands of jobs daily. Unfortunately for the early vendors – and their customers, it is much easier to design a product to be scalable from inception than to try to retrofit scalability into an older design.

4. **An understanding of true 24x7x365 operations.** When mainframe job schedulers were first created, there were two distinct periods of operation in the data center – online and batch. Typically during the day, the databases were 'online' and only available to end-users for true "real time" tasks that involved entering transactions at their terminals. At night, the databases were brought 'offline' and then batch activity, typically reporting and other batch intensive activities, were allowed to process. This batch window was often as long 12 hours (for example, 6:00 pm to 6:00 am). After the first generation of distributed scheduling products was built in the early 90's, people began to speak of the 'shrinking batch window;' however, for the new UNIX platform, the batch window was still very much intact.

   By the late-90's, when the latest generation of job scheduling products was being created, two significant issues had started to change the landscape for corporations and software providers alike, namely globalization and e-business. These phenomena began to overwhelm data center operations with the need for true 24x7x365 operations. The net effect was the virtual elimination of the batch window which essentially created a new set of requirements for scheduling tools. The latest generation of job schedulers is able to respond to these evolving requirements with the addition of new features for improved scalability and better, faster

workload management. Ultimately, these new requirements require the job scheduler to be a true "workload manager," responding to a wide variety of events for launching and managing jobs, and not merely relying on the traditional date and time model of scheduling jobs. A number of these features including event management are discussed later in this document.

Over the years, job scheduling has matured as a discipline and become accepted as a requirement for any comprehensive infrastructure management strategy. During this maturation phase, the industry has developed an understanding of what the common, or core, functions that a product needs to be considered a 'serious' or 'industrial strength' product. This section of the evaluation guide discusses this core set of features commonly acknowledged as requirements for any serious scheduling tool.



Business Calendars

At the heart of any scheduler is the concept of starting jobs at a particular date and time. Internally, businesses actually run on a variety of calendars including fiscal calendars, manufacturing calendars, payroll calendars, holiday calendars and others that all drive specific aspects of a modern business. Job schedulers are expected to provide comprehensive calendaring facilities that are easy to use, graphical in nature and able to combine calendars with one another to achieve a desired result (for instance, what day should payroll be run if it happens to fall on a holiday?) Surprisingly complex calendars can be required to manage a modern corporation with some companies managing several hundred depending on their size and number of geographic locations. Because of the vital nature of this feature, buyers are advised to scrutinize the calendar capability of any product to see that it meets their needs in terms of flexibility, ease of use and complexity.

Dependencies

Right after the concept of the business calendar in terms of importance, comes the idea of the 'dependency.' As the word implies, this is the ability of the scheduler to control the execution of the various tasks by having them execute not just on a certain date, but also in a particular order and in conjunction with other tasks. The simplest expression of a dependency is Job B follows Job A. In other words, Job B cannot execute unless Job A has completed. However, this simple concept can get very complex very quickly. Job B might have to wait for a particular file to arrive, or it might have to wait for some data to be input by a user, or it might be restricted from running after a certain time in the evening. When looking at the dependency capabilities of a product, it is important to have a clear idea of the types of dependencies that exist and to be able to easily map those to the product. If the product cannot create the calendar that is needed or run jobs in the correct order and with the right dependencies having been satisfied, then it is not likely to ever meet a site's core business needs and will be very cumbersome to use.

Auto Recovery

In a perfect world, jobs would run correctly every time; however, such is not the case. From corrupt data file to users entering erroneous data to programmers making mistakes, all of these anomalies can lead to late or incorrect processing. To deal with such unpredictable issues, sophisticated job schedulers are expected to accommodate automated recovery actions. This feature needs to be flexible enough to allow for a variety of responses to a 'failure.' For instance, some failures are not meaningful enough to stop subsequent processing, while others will corrupt downstream processing and result in other failures, inaccurate reports and other problems. The product should be able to take distinctly different actions based on the relative severity of a problem encountered.

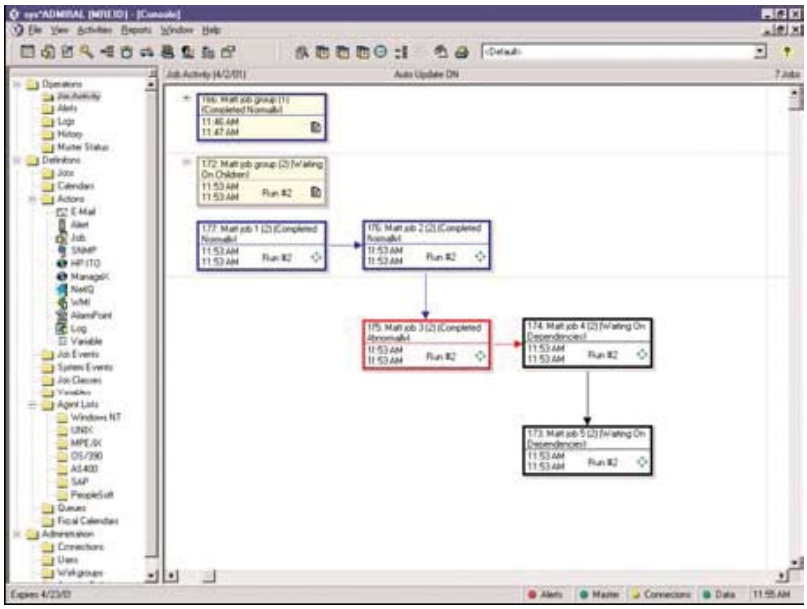
Additionally, the product should allow the user to create multiple types of recovery scenarios. For instance, in some cases it might be sufficient for the scheduler to simply stop processing altogether if the error is deemed severe enough. In other cases, it might be decided that when a specific type of error occurs the schedule should back up a couple of steps and rerun those jobs. If that is not sufficient, the user may run a series of related recovery actions like restoring a database prior to attempting to run the jobs again. Minimally, the product selected should allow the user to stop processing, continue processing and/or run a series of recovery actions before moving on to some subsequent step.

Alert Management

Because job schedulers perform such a vital role in the infrastructure, they must be able to generate an alert when something unusual happens with the processing. This alert management needs to be flexible enough to handle a wide array of potential events and also extensible so that it can identify and manage events of the users' choosing. Like auto recovery this feature needs to be able to respond differently to different types of events; in fact, alert management and auto recovery often need to work in conjunction with one another.

In a typical scenario, a job might fail, which in turn, initiates some type of recovery action. At the same time, there may be a desire to send notification of the failure to a designated person. This notification might be in the form of an email to a specific user, a page to a technician, or a message sent to a central management console. Additionally, this alert management should allow for some type of acknowledgement of the alert so that the scheduler itself is informed when the targeted person has received the alert. As with other features, ease of use and flexibility are key.

Some products say that they have this alert capability, but closer examination reveals that this is only possible if the user writes a variety of scripts to alert someone to a particular problem.

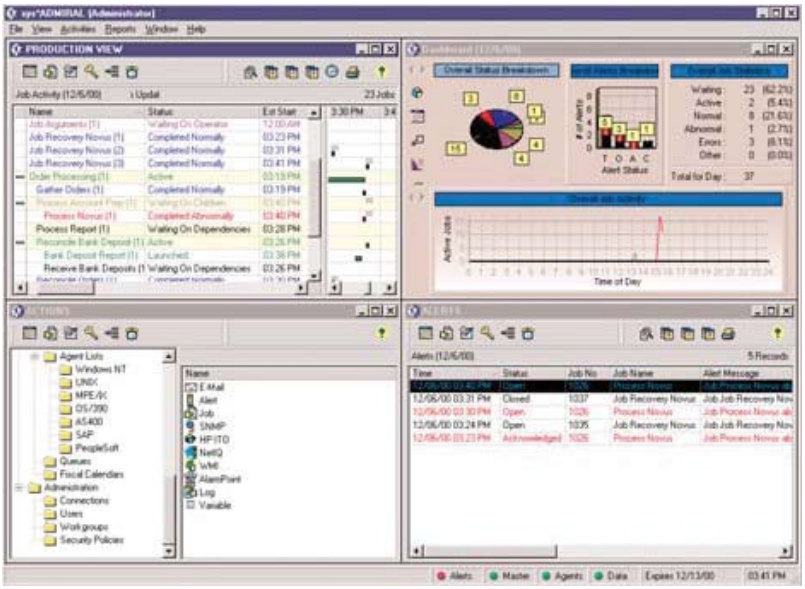


Tidal Enterprise Scheduler

Enterprise Application Support

The primary reason for implementing a job scheduling solution is to be able to support companies mission-critical applications. This requirement has existed since the early days of the mainframe and is still true today. What has become more complicated is the sophistication required to successfully integrate with today's leading applications. Many of today's leading applications – SAP, PeopleSoft, Oracle EBusiness, Siebel, etc., have some type of built in scheduler. At the same time, this scheduling capability is insufficient to manage all of the various applications in the enterprise, so integration of these core applications becomes a central issue.

Selected job scheduling vendors have extended their products to more readily encompass the needs of these critical applications and created interfaces that talk directly to the core scheduling functionality in these applications. If you have one of the applications listed above, there are commercially available interfaces available and it would be worthwhile for you to evaluate them. At the same time, not all interfaces are created equally, so take the time to understand how the vendor actually supports the interface, how current the interface is, what specific features are available to support the application and whether or not the application support is certified and on what level the certification was achieved.



Framework/Network Management Integration

Many companies today have implemented some type of network or systems management console. These products provide a variety of features and functions, but in many cases are implemented to provide a single-console view of the enterprise. This single console typically deals with the notion of "management by exception," which is simply the idea that given the incredible number of events that occur within a moderate to large IT shop, the operations personnel only want to deal with the exceptional conditions (typically the most serious errors).

Because a huge number of IT processes run in batch mode, it is critical that the scheduler integrate with the data center's chosen framework or network management console. Typically this integration will be twofold: First, and more obvious, the integration should allow the scheduler to inform the network console when there is some type of job failure (this is, in effect, another type of alert management as discussed above). Secondly, the integration should allow the network management tool to monitor the scheduler and its associated infrastructure. Although not as obvious, this integration gives the network tool the ability to monitor the health of the scheduler itself. In this way, if the scheduler or one of its distributed components should experience an error, the network management console should be able to report on any failures.

Security

It should be apparent that security is a vital requirement of a job scheduler. If your job scheduling product is in charge of running the mission-critical processes in your data center, then clearly you must control access to it. What may not be so obvious, is that in addition to controlling access to the scheduler itself, an administrator may also want to control access to individual features within the product by user or by group.

This requirement exists because of the diversity of users, not all of whom need access to all scheduling functions. For example, operations personnel are typically given broad access to the tool, but you might want to restrict their access to certain jobs or certain features within the scheduler. In some corporations, end users are

given limited access to the product so that they can monitor the progress of jobs of particular interest to them.

Other personnel may have the authorization to create certain business calendars or jobs, but they do not have the ability to run those jobs in production.

The key when looking at the scheduler's security features is to look at ease of use and granularity. Ease of use is necessary for quick authorization changes for a given user or group of users in response to changing business needs. It is dangerously shortsighted to compromise an operation's security policies simply because it is deemed too difficult to implement a particular policy. Granularity, or refinement, is important because of the need to make only certain features of the product available to certain users. Granularity makes it possible to easily grant precisely the types of user rights to just those users who need them.

Audit Trails

Many people relate audit trails to security and while there is a strong connection, this is not the only benefit of audit trails. With audit trails in place, operations personnel can monitor and, when necessary, undo changes to the scheduling environment. For instance, even authorized and well-intentioned users can make mistakes when modifying the production schedule. Audit trails the means to understand precisely what changes were made to the production environment, and who made them. Given the rapidly changing requirements in data centers, it is important that all changes to the production environment are recorded.

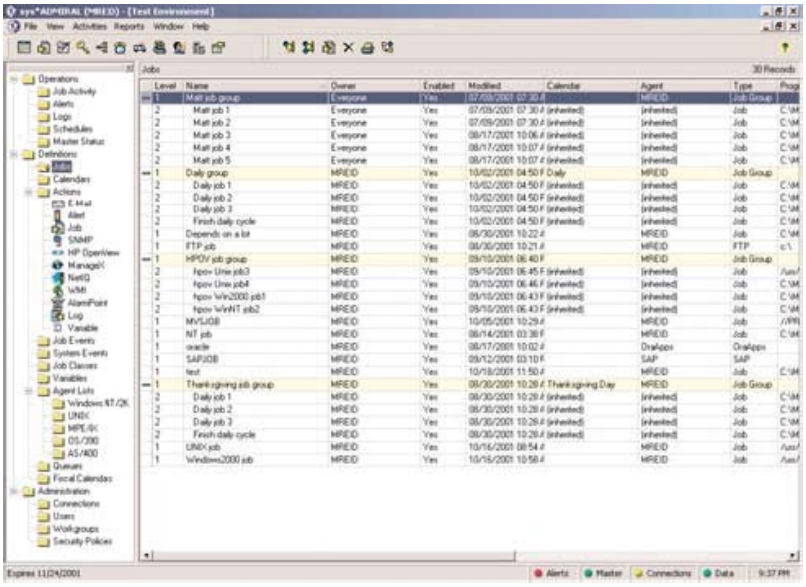
A related topic is the concept of system or scheduling logs. While true audit trails are typically focused on what changes were made, by whom, and when, logs are simply the documentation about the results of a particular execution of a job or schedule. Well-organized logs give the user a clear indication of exactly what workload ran and when it ran.

Ease of Use

Although this feature is hard to describe, most users feel "they know it when they see it." More accurately, they tend to know it when they experience it. Although this might be viewed as a 'soft' requirement that is hard to pin down, it should not be underestimated. Personnel change jobs frequently and we often don't have our most experienced personnel on site at the precise moment when a failure occurs or the daily schedule needs modifying. For most users the familiar "Windows Explorer" interface is the most intuitive of all; newer products have generally adhered to this type of interface, and, in some cases, added additional graphical capabilities to further simplify usability. It is also important to point out that tools with intuitive interfaces are not only easier to use, but ultimately produce fewer user errors and faster recovery when errors do occur.

Ease of Installation and Configuration

Given the dynamic nature of the modern data center, with new hardware and applications being added almost continuously, it is important that a product be simple to implement and reconfigure.



During the evaluation phase, it can be difficult to assess configuration and reconfiguration, but potential buyers are encouraged to scrutinize, at least the initial installation of the product. Can you do the installation yourself? If not,

can you at least follow along with the vendor to understand the installation process? How long did the implementation take? Hours? Days? Any product that takes more than a day to get a basic implementation up and running will probably not be any easier to reconfigure when the operating environment changes and, hence, is not a good candidate for anyone who anticipates changes to their environment.

Queues and Prioritization

The concept of queues dates back to the mainframe and is very simple: not all jobs and schedules are created equal. Not all jobs have the same priority or requirements. A queue is generally used to control resource allocation by the scheduler. For instance, a queue can be created that only supports lower-priority jobs. All jobs of a certain priority or below get placed in that queue, which is set to a lower priority than some other queue where more important workload is being managed. A classic example of using queues is to have all user-submitted jobs put in one queue for execution during off peak times, while regularly scheduled production jobs are placed in another queue of a higher priority. In the area of queues and priorities, a scheduling product needs to support the number and type of queues and prioritization schemes required to effectively ensure that high-priority jobs finish before the lower-priority workload.

Architecture

Although every vendor describes its environment differently, the accepted approach to distributed scheduling is to have one or more central processing hubs – often called a 'master' – and then a variety of distributed components that are installed on any distributed machine where jobs need to be processed – typically called 'agents'. The last required piece of the architecture is the client or GUI. It is generally accepted practice to separate the GUI layer from the core product, as it tends to make the product easier to configure and maintain.

A key requirement to assess in this area is the amount of flexibility and fault tolerance available. Flexibility is really a measure of how easy the product is to deploy in a given environment. Can you have the master on your machine of choice? Can you have the agents and/or application adapters deployed on the most desirable machines? Can you run more than one master scheduling node and have the various masters interacting with one another in an intelligent way to provide increased flexibility or fault tolerance? Some vendors have attempted to disburse scheduling data throughout the environment to make each scheduling node the equivalent of the 'master', but the resultant issues associated with error recovery in the event of a node or network failure have largely ruled out this type of architecture. Since the vast majority of jobs described in a network have dependencies on other jobs on other machines, there is little justification for the added complication of moving scheduling data around the network, when it can be maintained more reliably and securely at a central location.

Fault Tolerance

Fault tolerance as it relates to scheduling is usually a reference to providing a 'hot' or standby master or central processing hub. This secondary master is configured so that it will immediately take control of the scheduling environment in the event of a failure by the original master. Over the years, vendors have tried a variety of approaches to fault tolerance, but the generally accepted approach is to have a third node in the configuration that monitors the health of the master. If this third node loses contact with the master for some predetermined period of time, then primary responsibility for processing is handed off to the secondary or backup master, so it can continue to manage processing until instructed to do otherwise.

In the early days of distributed technology the primary cause of computing failure in the enterprise was often a network failure. Some of the first generation vendors experimented with 'network fault tolerance,' or 'cooperative computing' models to deal with network failures. In this model, each node/machine in the scheduling environment has essentially a complete scheduler installed on it. This model typically leads to a heavier and more complex application being installed at each scheduling node and also requires that scheduling data be moved around the network to each machine having one of these additional scheduling engines. This continual movement of data around the

network can cause problems during failures since each node in the network may need to be recovered individually.

Graphical Management

In today's world, it is a requirement that scheduling products provide true graphical management of the environment. This graphical management can take a variety of forms, but should include an easy to understand central console from which all jobs and schedules can be viewed. This list of jobs should be filterable so that only certain types of jobs are visible or can be sorted to the top of the list. Additionally, the console needs to be color-coded so that certain jobs – failures, in particular – can be highlighted for easy viewing. As the console gets more sophisticated, it might have other indicators to tell users at a glance if one or more jobs are in an error state.

More sophisticated products have 'dashboard'-like features which present the ongoing results of scheduled activity in truly graphical format such as bar charts. These charts can give novice and experienced users alike a quick reference to the progress of the daily schedule and the number of jobs that have finished successfully/unsuccessfully, the percentage of the daily schedule that has already executed, and other common measurements.

In addition to easy to use consoles, modern schedulers have true graphical capabilities that represent job streams graphically to make it easier to understand job status and interdependencies. These pictorial representations make both the creation and trouble shooting of job streams significantly easier.

Platform Coverage

Because a typical distributed enterprise includes more than one type of hardware and operating system, it is a requirement to thoroughly analyze current and future needs with regard to the scheduler's platform and technology support. A vendor should support the data center's current and future operating system requirements and have a history of staying current when the operating system is updated. It is also important to consider the data center's historic and projected preferences for computing technology. For instance, although Windows is the clear choice for the GUI, do you want your distributed job scheduling 'engine' or 'master' to reside on UNIX or Windows? Ideally, being positioned to have both UNIX and Windows is the best strategy, since needs may change over time. An OS/390 data center will likely want the flexibility to have a truly integrated enterprise solution that provides integration with their mainframe scheduling solution.

Database Support

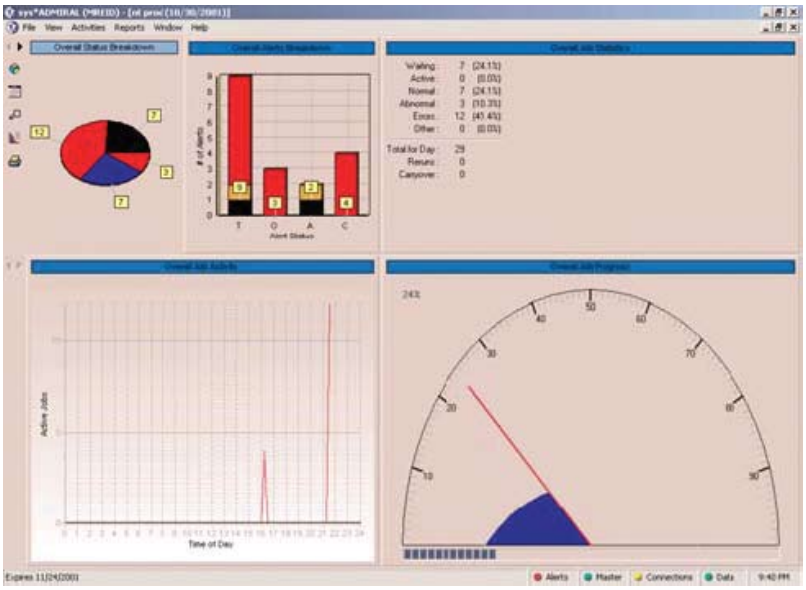
Another aspect of platform coverage is database support. On the mainframe it was and is commonplace to use 'flat' or indexed files as the underlying data repository for all of the scheduling objects and rules. At the time these products were written, relational databases were not in wide use; in fact, DB2, now the leading mainframe database product, didn't arrive until the late 1980s. Early distributed scheduling tools followed the flat-file model, because they lacked reliable relational database technology. However, with the arrival of robust reliable relational database technology in the middle 1990s, all of the leading products now use a relational database as their repository. Another benefit of using relational technology is the ability to mirror the databases if desired and incorporate this into a fault tolerance or backup strategy, further facilitating unattended operations.

Most of the job scheduling requirements listed are considered core requirements for a modern distributed scheduling solution. Some of these requirements have evolved over time, like the need for a relational database, graphical management and a Windows-based interface, while others, like support for business calendars and dependencies have remained fairly static over time.

At the same time, there are some new requirements that have emerged recently that represent the 'state of the art' in job scheduling. These new requirements include:

- Event-driven processing
- Event adapters
- Comprehensive APIs
- Scalability
- Web-based interface

Tidal Enterprise Scheduler



Event-driven Processing

When job schedulers were first created, the clear methodology for managing jobs revolved around time and dependencies. Job schedulers were developed to run jobs at the right time and in the right order. While those requirements still exist, there are also new requirements for managing 'jobs' on a real-time or event-driven basis.

The adoption of Java and .NET for hosting applications has sparked a need to manage batch transactions that spawn across multiple platforms to fulfill the complete business process; however, neither application possesses scheduling features. In order to support these modern types of applications an effective job scheduler must accommodate the scheduling needs of both near-time and batch tasks across a diverse set of infrastructure platforms.

With an event-driven approach, the scheduling tool does not continually 'ask' (poll) if an event has taken place, but instead, the scheduler can be 'told' immediately when the event has taken place by working closely with the operating system – for example, a file arrival. There are hundreds of other events that can be used to trigger jobs – changing data in a database, network events, system events (such as memory utilization or disk utilization) and so forth.

Additionally, if the job scheduler is well integrated into the environment (see section on API's), the scheduler can also process events coming from other systems management products like network managers.

Event Adapters

Today's increasingly complex and dynamic IT environments require that systems be managed much more holistically than before. The increasing need for speed and efficiency dictates that all of the applications, databases, systems and networks be managed not as individual components, but as a single integrated environment. Job schedulers are a core technology for addressing this requirement, but to do so they must be able to communicate rapidly and reliably with the other applications and technology components in the environment.

To facilitate this rapid communication, the job scheduling vendor must supply the required event adapters that integrate with the wide variety of applications and systems in a given environment to capture the events necessary for efficient and effective management. If a specific event adapter is not available, then other means for integrating with the scheduler should be available so that custom solutions can be created.

Comprehensive Programming Interfaces (API's)

In traditional job scheduling, the scheduler was always the controlling entity and executed or invoked other applications in the environment. In today's more complex environment, it is often desirable for other applications to interface with (or 'call') the scheduler directly in order to have the scheduler provide scheduling services on application's behalf. Newer schedulers provide comprehensive programming interfaces that easily facilitate integration with other applications.

Scalability

Today's job schedulers are required to be vastly more scalable than their first generation counterparts. It is not uncommon for mainframe schedulers to manage 100,000 jobs per day. In distributed job scheduling, most companies still manage a few hundred to a few thousand jobs per day. More computingintensive companies are beginning to manage in excess of tens of thousands of jobs per day. Many of these companies are finding that their first generation schedulers are unable to keep up with their increasing workloads. The requirement for increasing scalability will continue to escalate as corporations begin to manage more jobs and also incorporate more event drive capabilities into their scheduling requirements.

Web-based Interface

In today's world it is standard practice that applications be "web enabled" and job schedulers are no exception. A web interface for a job scheduling tool simplifies administration of the tool itself and also gives operations personnel the utmost in flexibility to log on from anywhere to monitor and control their scheduling environment. When looking at the web-based capabilities of the product it is also important to know just how much of the product is available through the web since some tools have only limited capabilities in this area.

Conclusion

A job scheduler is a core component of an enterprises' overall systems management strategy. With job scheduling, enterprises attempt to proactively manage the flow of essential business processes that are required to run the business. The core features of job scheduling are well understood and have stood the test of time. At the same time, each vendor has a slightly different interpretation of these core features. Additionally, the market continues to evolve and requires new features, continuous innovation and new application support. Enterprises need to continually evaluate their applications management strategy to determine their job scheduling requirements and whether their current solution meets their current and future needs.

TIDAL®
SOFTWARE

2100 Geng Road, Suite 210, Palo Alto, CA 94303

1 (877) 55-TIDAL
info@tidalsoftware.com

www.tidalsoftware.com

1012-A 06/06