EMC® Documentum®
xCelerated Composition Platform

Version 1.0

Performance Tuning Guide

EMC Corporation
Corporate Headquarters:
Hopkinton, MA 01748-9103
1-508-435-1000
www.EMC.com

Table of Contents

List of Figures

List of Tables

Preface

This document addresses areas known to affect performance in xCelerated Composition Platform (xCP) solutions. The guidelines can be used during the development and testing of new xCP solutions or to troubleshoot performance problems with production xCP solutions.

The main body of the document is organized around the main areas impacting performance. Many of the individual topics, however, were derived as solutions to the most common performance problems know to occur in production systems. These known problems are listed in the troubleshooting chapter and include links to the topics that address the problem area.

While system sizing has a significant impact on the performance of your tuned solution, it is only addressed in summary fashion here.

Intended Audience

This document provides the lowest level of aggregated technical detail available on the subject of xCP performance tuning to date. Designers of xCP solutions comprise the primary target audience for this document, though others can have an interest in its contents as well.

Revision History

The following changes have been made to this document.

Revision Date	Description
April 2010	Initial publication

Chapter 1
Overview

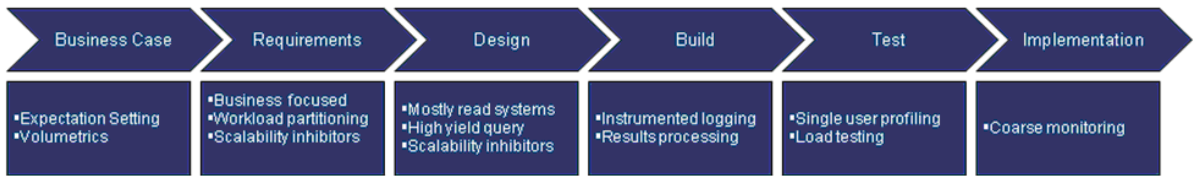
Creating the best performing xCelerated Composition Platform (xCP) systems to address your business requirements involves; 1) designing and configuring the software to maximize throughput and minimize response time, and 2) sizing your hardware components. This document focuses on designing and configuring the software. Chapter 2, Planning Capacity and Sizing provides summary information on capacity planning and system sizing. The following chapters provide detailed information on designing and configuring high performing xCP systems:

- Chapter 3, Maximizing Process Throughput
- Chapter 4, Designing the Application
- Chapter 5, Designing Reports
- Chapter 6, Configuring System Components
- Chapter 7, Measuring Performance
- Chapter 8, Maintaining the Repository and BAM Databases
- Chapter 9, Troubleshooting

Software development lifecycle

Figure 1, page 11 illustrates a typical Software Development Life Cycle (SDLC). Performance tuning opportunities exist at every stage of the development process.

Figure 1. Performance tuning software development lifecycle



Business case — Expectation setting and volume metrics characterize the business case phase. Make sure that realistic system expectations are defined up front. For example, it is unrealistic to expect being able to fetch 5 TB digital movie master file in 2 minutes. Carefully define current and future business transaction volumes as they have a significant impact on capacity planning.

Requirements — During the requirements phase, focus on defining the business requirements, not the implementation details or functional requirements. For example, collecting tax is a business requirement while having a tax lookup table is a functional requirement. Define the workload partitioning scheme in accordance with business process requirements. For example, tax returns are filed and processed in different locations depending on where you live. Remove requirements that inhibit scalability, including case insensitive partial keyword search, excess search fields, large search result sets, and so on.

Design — Most transactional systems are characterized as read-only (90% of transactions are read-only). Exceptions to this rule include report generating systems, like billing statements, which are mostly write systems. Most systems have a few business transactions that dominate system load. Design the system to optimize performance for high load transactions. Exceptions to this rule include infrequently run business transactions, like a once-a-year audit committee report that determines business funding. Analyze your queries for scalability inhibitors.

Build — Instrument logging so it can be turned on or off. Assess query results and consolidate multiple fetches (multiple queries) into single queries.

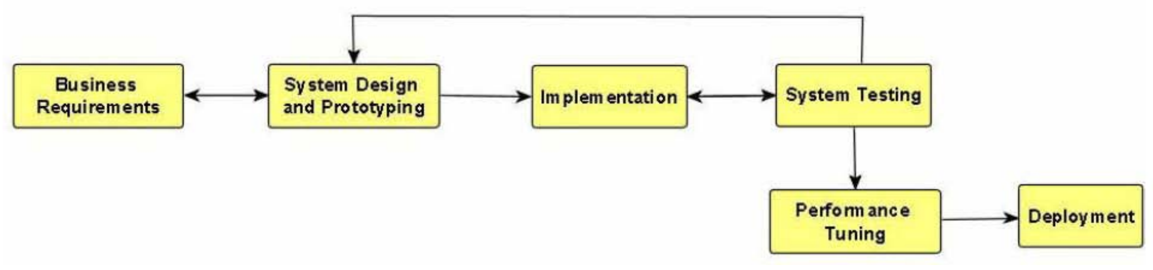
Test — Assess performance for single user tests and system under load. Use scripted business transactions for single user testing. For load testing, turn off detailed tracing, use reasonable click pacing, and use the same scripted business transactions as used for single user testing.

Implementation — Monitor coarse system utilization metrics (CPU, memory, disk I/O) correlated with content, users, and business transaction volume. Maintain the databases.

Using iterative development

Create your xCP application in a series of phases. Break up the full application into modules and establish frequent cycles of design, implementation, and testing (Figure 2, page 12). For xCP applications, the iterative approach is more successful than the classic waterfall approach.

Figure 2. Iterative performance improvement



Thorough planning and testing is important for process-based applications and can make a difference in whether a project succeeds or fails. xCP provides templates for fast prototyping and testing of processes, forms, reports, and user interfaces. Anticipating risks, aligning on requirements, and designing the solution carefully are critical to success.

General guidelines

Pay close attention to the following general performance guidelines:

- Tune the software before completing your sizing requirements (Tuning the software first, page 13).
- Design your application to enable users to complete their jobs with a minimum number of actions and choices.
- Audit, monitor, and report on necessary information only.
- Partition the workload for balance and avoid bottlenecks (Balancing system load, page 13).
- Minimize the number of discrete object types through use of structured datatypes and avoid excessive subtyping in your object model (Designing the process object model (using structured datatypes), page 37).
- Tune your queries for the highest yield (Maximizing query yield, page 36).
- Maintain your databases (Chapter 8, Maintaining the Repository and BAM Databases).
- Avoid sharing resources with other applications (Avoiding resource sharing, page 14).

Tuning the software first

Use the guidelines provided in this document to tune and configure the software first, before completing your hardware requirements assessment. After tuning the software, simulate the activities and load of your anticipated production environment to assess hardware resource needs.

In most cases, do not compensate for software design issues by adding hardware resources. Not only do hardware resources cost more, but resource requirements stemming from poor software design and maintenance practices can gradually erode and overwhelm hardware capacity.

Balancing system load

Load balanced systems provide the best overall performance. A large part of system load balancing involves finding the right balance of software components to handle system activities. After determining the correct balance of software components in a simulated environment, you can more easily determine the most appropriate deployment of those software components to the same or different hosts (physical or virtual) in a production environment.

Use an iterative methodology to determine the combination of software components that result in the most balanced loads. Run the software with loads that simulate the anticipated production environment and measure activities between the key software components where bottlenecks can occur. When bottlenecks occur, try to reconfigure or change your design first, before adding additional servers or hosts. Add more hosts only after you have exhausted all software options.

Note: Adding more hosts before you tune the software only masks problems in your design. In some cases, no amount of additional hosts solves a software or implementation design issue.

Virtual machines provide a convenient way to simulate the effect of adding additional hosts to an environment. Virtual machines can be set up and torn down easily and they can be configured with their own dedicated resources. You can also use tools, like Shunra, to simulate network latency between different host machine locations in your production environment. Virtual machines do not simulate the capacity of physical hardware exactly, but they can be used to gauge the effect of adding hosts of different types. As such, they can help you decide on the number and type of physical hosts to handle the throughput requirements of your business.

Addressing bottlenecks

Focus your load balancing strategy on system areas prone to bottlenecks. Devising good measurements of throughput for these areas helps you iteratively change aspects of your software design and assess the impact on throughput. After tuning the software, you can add hosts (virtual or physical) to bottleneck areas that cannot be designed away. Key process areas prone to bottlenecks include the following:

- automatic and manual activity creation
- manual and automatic activity completion
- work queue and audit trail management in the repository database
- system query performance.

Avoiding resource sharing

When possible, provide dedicated hardware resources for xCP system components. System performance fluctuates when one or more tiers in the system share resources with other applications.

Partitioning of xCP resources opens the possibility of resources not being available to xCP when needed. This effect also occurs when running virtualized images on a large host machine. Bottlenecks that occur due to disk I/O can be hard to diagnose. CPU and memory utilization do not indicate which application or image generates the I/O.

Minimize the use of multiple schemas on your database host and do not use the same database instance for your repository and for BAM.

Dedicate each JVM or application server cluster to a single application, such as TaskSpace. Dedicated JVMs enable applications to scale and perform better, as well as making them easier to troubleshoot.

Chapter 2

Planning Capacity and Sizing

This chapter provides the following capacity and sizing related sections:

The relationship between content (number of documents and average content size), flow (content ingested per time period), retention policies (content disposition rates), and growth projections affect your capacity planning. Plan your capacity to accommodate future content and flow. A small system comprised of a repository with 10 million objects and 2 million activities per day can grow to a large repository with 10 billion objects and 20 million activities per day.

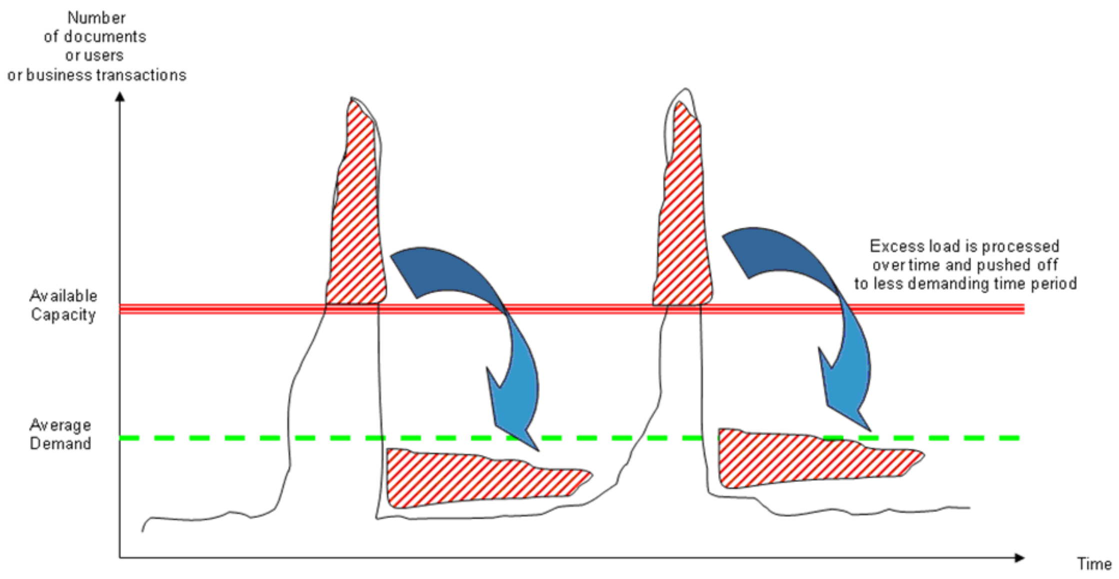
Plan your capacity to account for content disposition rates and auditability. Solutions that grow to 1 million documents by ingesting 100,000 documents per year for ten years has a different impact from solutions that ingest and dispose of 1 million documents per day. Solutions that retain an audit trail for a long time have a significant impact, even if the daily workflow volume is small. After ten years, a solution which averages only 50 workflows per day can generate an audit trail database table with 100 million rows.

Planning for peak or average loads

Systems with high transaction volume or large usage (100,000 activities per hour, for example) can exhibit variable performance during certain time periods. Sizing the system for average rather than peak loads can result in degraded performance during peak load periods. For mission critical applications where downtime or performance degradation is unacceptable, size the system for peak loads.

Gather all use case scenarios during the requirements phase. If there are periods when workloads spike (Figure 3, page 16), make sure to identify and use these volume metrics when sizing your system. When load testing, include single user, average-load, and peak-load scenarios.

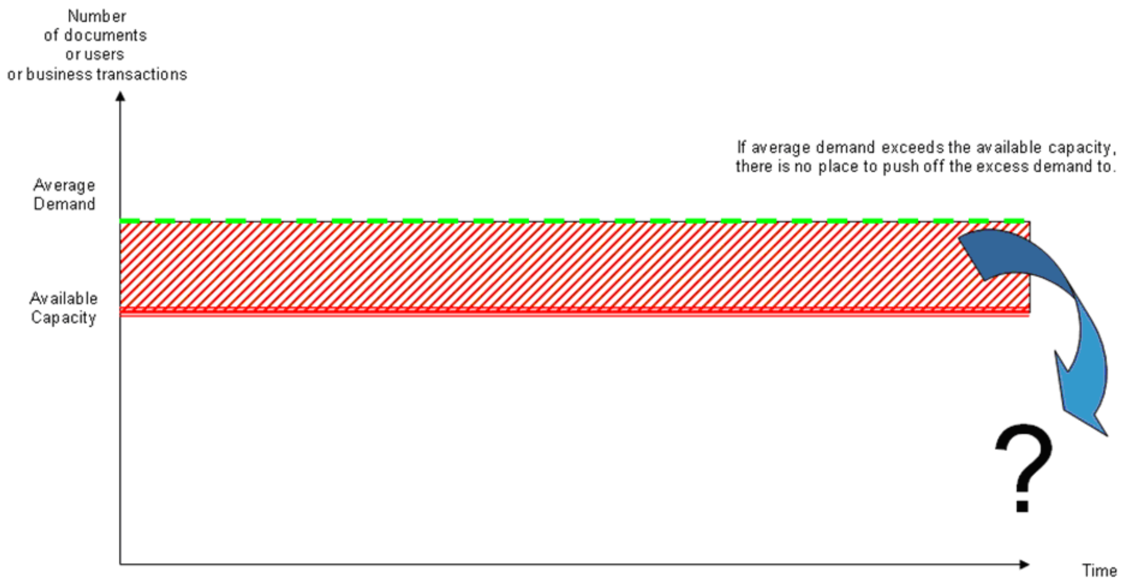
Figure 3. Peak versus average loads


Number of documents or users or business transactions
Available Capacity
Average Demand
Excess load is processed over time and pushed off to less demanding time period
Time

Lack of capacity to handle peak loads does not matter if you can defer the processing of peak periods over a longer period when volumes are less than available capacity. However, process deferral can increase cycle time and violate your SLA.

Size your capacity (maximum available throughput) to handle peak loads within specified SLA. If your SLAs are relaxed enough to be able to defer processing of peak loads, you can size capacity to a demand curve that fits or exceeds the average load (Figure 4, page 16).

Figure 4. Average must be within capacity

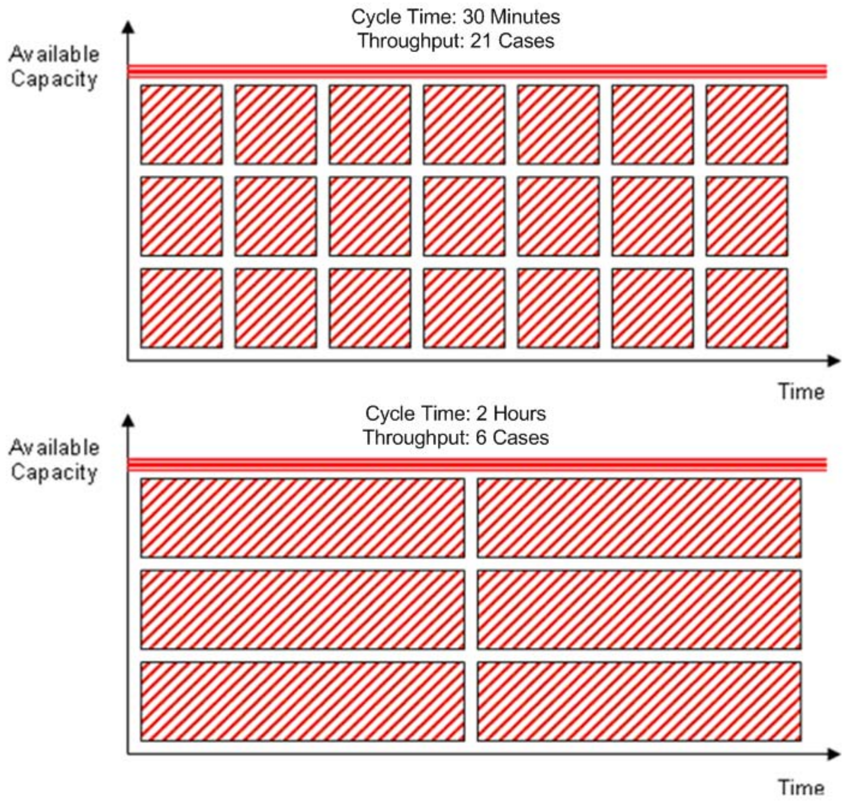

Number of documents or users or business transactions
Average Demand
Available Capacity
If average demand exceeds the available capacity, there is no place to push off the excess demand to.
?
Time

If your average demand exceeds your available capacity, you cannot defer processing to another time period. Capacity that is less than average demand leads to a system that can never catch up with the backlog.

Planning for workflow database tables

Workflow cycle times and the number of activities per workflow impacts sizing of supporting workflow database tables (dmi_workitem, dm_queue_item, dmi_package, dmc_wfsd_*). Figure 5, page 17 illustrates the difference between a system with 100K workflows, each with 10 activities and a cycle time of 1 day, and a system with same number of workflows but with 100 activities per workflow and cycle times of 365 days. The system with cycle times of 1 day generates no more than 1 million rows in the workflow supporting tables (dmi_workitem, for example) whereas the system with cycle times of 365 days generates more than 3.6 billion rows in the workflow supporting tables. When sizing your supporting workflow database tables, consider not only the number of workflows and activities, but the cycle times as well.

Figure 5. Cycle time versus throughput



Characterizing content

The relationship between content and capacity planning differs depending on whether content is taken to mean the number of documents or size of the documents. For example, two clients can both have a 500 TB of content. In one case, 10 billion emails (50 KB each) comprise the content whereas in another case 100 master files (5 TB movie productions in high definition format) comprise the content. In the former, the potential flow of repository objects and process activities related to the 500 TB of content is much greater.

Capacity planning worksheet

Use the information in Table 1, page 18 to help with your capacity planning.

Table 1. Capacity planning worksheet

Load source	Question	Reason	Sample answer
Content	How long are documents retained after workflow completion?	Document retention requirements determine disposition rates and impact future capacity requirements.	We keep all documents for at least 33 months. 20% of documents are held permanently.
	What are the average, peak, and bucket cycle times for your workflows?	Workflow cycle times determine which cases are hot and which are cold, and allow for case tiering and SLA adjustments.	95% of all cases complete in less than 5 day. 99% of all cases complete in less than 3 month.

Load source	Question	Reason	Sample answer
Flow	What are your peak and average daily document ingestion rates?	An influx of documents can cause an SLA exception unless the system is sized for peak ingestion.	At 5 year horizon, we project ingestion of 345,000 documents per day. This may peak to 1,000,000 documents per day during a single day in a year.
	What are your peak and average workflow starts per day?	An influx of workflows can cause an SLA exception unless the system is sized for peak workflow starts.	At 5 year horizon, we project starting 191,000 workflows per day. This can peak to 382,000 workflows per day during a single day.
	What are your peak and average number of documents attached to a workflow?	Document volume affects storage capacity requirements and determines size of the tables that track attachments.	At 5 year horizon, we project an average of 2 documents per workflow. This can peak to 5 documents per workflows during a single day in a year.
	How many peak and average audit trails do you have per document and workflows?	Audit trails determine how many rows are in the tables that track events.	We think we have average of 10 comments per document and 2 comments per workflow. Some of the very difficult cases have 100 comments per document and 20 comments per workflow.
	What is the average number of document views you have per document during workflow processing and over the document lifetime?	Document views determine the retrieval rate and probability of retrieval after case closure. Document views can affect your tiered storage approach and demand for network bandwidth.	We have 3 document views per document during workflow processing. There is total of 3.1 document views over document lifetime.

Load source	Question	Reason	Sample answer
User	What is the peak number of concurrently logged in users during a busy hour?	Feeds directly into the sizing spreadsheet.	There are 2500 concurrently logged in during peak busy hour.
	What is the average number of business transactions per user during a busy hour?	Used to adjust the workload modeled in the sizing spreadsheet.	There are 10 business transactions per user during a busy hour.
	What is the average business transaction duration per second during a busy hour?	Used to calculate the percentage of active users.	During a busy hour, the average transaction lasts 60 seconds and there are 10 transactions. The percentage of time users are active is:10*60/3600=16% active
	What is the average number of hours a worker works in a time period?	Determines available worker capacity to process cases, based on average case processing time.	On average, a worker works 7.5 hours per day, 260 days in a year. If one case takes 15 minutes, one worker can process 4 cases per hour and 7,800 cases per year (4*7.5*260=7,800). 2,500 users can process 19.5 million cases per year.
Workload	What are the peak and average number of business transactions (cases, requests, and so on) per time period?	Determines the size of the workflow activity tables.	We process 10,000 cases per month, but occasionally we get a seasonal influx of 100,000 cases per month.
	What are the peak and average cycle time of business transactions (cases, requests, and so on)?	Determines the size of workflow activity tables.	Average is 5 days but some cases can stay open as long as 90 days.

Load source	Question	Reason	Sample answer
Search	What are the top three or four search patterns on the system?	Helps you decide how to create targeted search forms, with fewer fields, so that the supporting database is indexable.*	• For person search, three search fields are required. They are social security number (required), date of service (required), and document type (optional). • For company search, three search fields are required. They are plan number (required), date of service (required), and process area (optional). • For medical search, three search fields are required. They are provider number (required), date of service (required), and patient ID (optional). • For case search, two search fields are required. They are case number (required) and date of service.
Search	What is the distribution of the time periods for the search in terms of number of searches?	For large volume systems, this provides a DBA with an idea of which data is hot and which is cold, which helps with database design partitioning.	• 60% of searches are against documents over past 1 month. • 25% of searches are against documents over past 3 months. • 7% of searches are against documents over past 12 months. • 2% of searches are against documents over past 24 months. • 1% of searches are against documents over past 36 months.

Load source	Question	Reason	Sample answer
Search	What are the top three or four highest yield workload profile use cases that cover 80% of business transactions?	Provides capacity sizing information such as activities per second based on the underlying workflow template design.	• 40% of the time – straight through processing. Case processor reviews inbox artifacts and terminates case. • 20% of the time – straight through processing with research. Case processor reviews inbox artifacts, searches for previously submitted documents, and terminates case. • 10% of the time – expert review. Case processor reviews inbox artifacts and routes case to expert for review and disposition. Expert uses straight through processing to terminate case. • 10% of the time – medical review. Case processor reviews inbox artifacts and routes to medical processor for further review. Medical reviewer determines disposition of case and forwards back to case processor. Case processor terminates case.
* No database can handle the number of indexes required to search with any combination of the 7 fields under a reasonable volume with reasonable response SLAs. This kind of "search pattern" classification allows proper database indexing that can scale to billions of records with proper database sizing, configuration, and maintenance.			

Sizing the system

Process-based applications with high volumes of activities have special sizing requirements in a production environment. Large process volumes can also raise issues of high availability and disaster recovery, which require more complex and robust hardware requirements.

Use the *System Sizing Spreadsheet - EMC Documentum 6.5* (available on Powerlink) and *BAM Dashboard Sizing Calculator* (available in the BAM Database Sizing Calculator folder of the bam-product-supplemental_files.zip file) as a starting point to estimate hardware requirements. Before using either of these sizing tools, identify and model all parameters affecting sizing that are identified in the capacity planning worksheet (Capacity planning worksheet, page 18).

The *EMC Documentum System Planning Guide* provides information on system topologies (distributed environments, high availability, and so on), which can also affect performance.

Begin system sizing during the design phase, since volumetric considerations influence the system design. For example, some high yield process transactions like the task list must be performed frequently. The task list transaction can perform well in a test environment, but can result in performance problems when many users simultaneously execute the transaction. Each transaction queries the database that must return results in under one second. If the browser is unable to render screens in a timely fashion, aggregate demand on the database adversely affects user performance. If there are not enough resources to accommodate thousands of simultaneous users querying the database, performance degrades.

Chapter 3

Maximizing Process Throughput

This chapter contains information on the following topics:

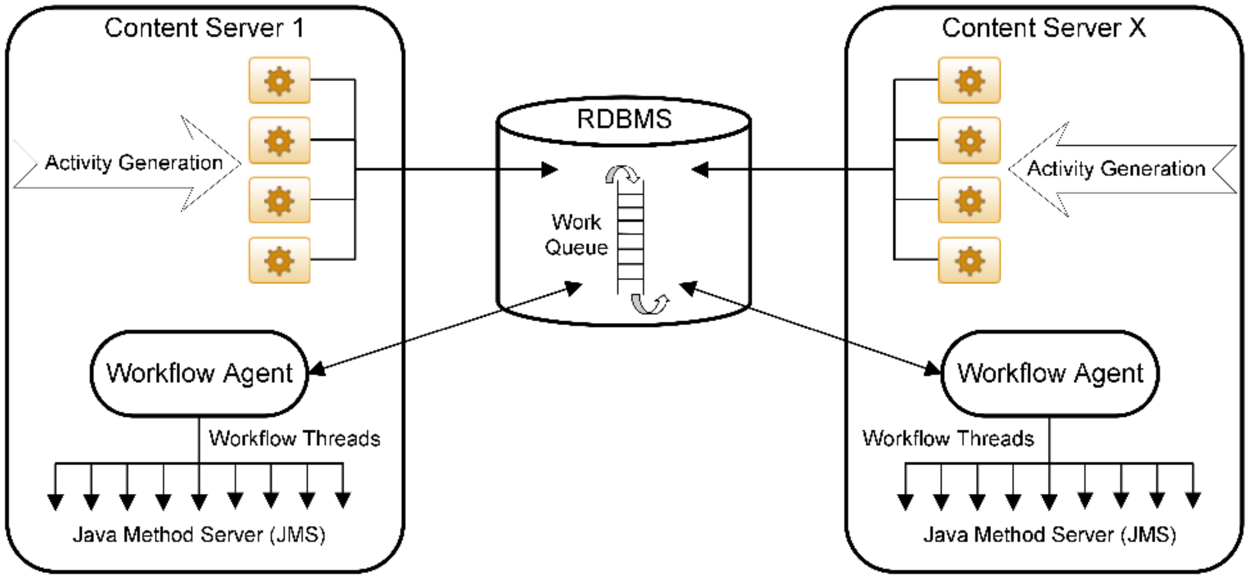
Understanding workflow throughput

Process Builder defines the automatic and manual activities comprising your business process. The frequency with which these activities complete determines your overall throughput. The following items impact process activity throughput:

- process activity creation rate
- database capacity
- number of Content Servers (workflow agents) processing activities in a work queue
- number of workflow threads for each Content Server
- TaskSpace performance
- process activity completion rate.

When a workflow instance creates an activity, the Content Server workflow engine adds a work item to a workflow queue in the database (Figure 6, page 26). Each Content Server in the system writes to the same workflow queue. The workflow agent for each Content Server then queries (polls) the database for work assignments from the common queue and adds the work items to workflow threads for the Java Method Server (JMS) to process. Drawing work assignments from the common workflow queue balances activity completion load across all workflow agents and Content Servers in the system, even though some Content Servers generate more activities than others. Understanding activity completion, page 27 provides details related to workflow agent processing.

Figure 6. Workflow throughput



Independent factors determine a systems capacity for generating activities and completing activities. The activities defined for all active workflow instances across all Content Servers determine the activity generation rate. The number of workflow threads available across all Content Servers provides a good indication of activity completion capacity.

In fully load-balanced systems, activity creation rates approximate activity completion rates. When activity creation rates exceed activity completion rates, the database work queue swells, which affects database performance. When the activity completion rate exceeds activity creation rate, system resources for activity completion become under utilized. Variations in the rate of activity creation cause peak load demands for activity completion.

Design your processes in accordance with your business requirements, then scale the Content Server tier to keep up with the activity creation rate during peak load periods (Planning for peak or average loads, page 15). If you are constrained in your ability to scale the system, you can also regulate the rate of activity creation (in your process design) so that it does not exceed activity completion capacity.

Assessing activity completion rate, page 27 provides information on the activity processing rate of a single workflow thread. Assessing activity creation rate, page 26 provides information on modeling the activity creation rate for all your processes over time.

To improve system capacity to complete activities, use more workflow threads per Content Server and add more Content Servers (Increasing workflow threads and adding Content Servers, page 28).

Be explicit about your response time and throughput requirements.

Assessing activity creation rate

Business requirements define the activity creation rates for the system.

Minimizing and consolidating activities

System throughput varies between 3-100 activities per second, depending on system configuration and hardware. Workflows with more activities take longer to complete.

The largest performance impact for processing activities results from opening up a new Content Server session. As a result, the biggest performance improvement comes from minimizing the number of discrete activities in a workflow. Minimize the number of workflow activities by, 1) eliminating unnecessary activities altogether or 2) consolidating the steps performed by multiple activities, into a single condensed activity.

To improve the completion rate of individual activities, do the following:

- Use the bpm_noop template wherever possible. This particular noop does not create an additional template and does not send an HTTP post to the JMS.
- Within the automatic activity, do the work on behalf of a superuser instead of a regular user.
- Turn off auditing whenever unnecessary.

Assessing activity completion rate

A single workflow thread can process up to five activities per second (using the bpm_noop activity template), which works out to be 300 per minute and 18,000 per hour (depending on the specific activity and method being called).

Understanding activity completion

Each Content Server provides one workflow agent to process workflow activities and one Java Method Server (or Docbasic Method Server) to support up to 25 (nominally 15) workflow threads.

The workflow agent polls the workflow queue for available activities (activities where the `a_wq_name` attribute of the activity has not yet been marked for processing). The workflow agent acquires available tasks from the queue and updates the `a_wq_name` attribute of the activity to mark the activity for processing.

The workflow agent acquires 30 activities (if there are 30 activities in the queue) for every workflow thread. For example, if Content Server provides three workflow threads, the workflow agent picks up 30*3=90 activities. The workflow agent provides the 30 acquired activities to each workflow thread for processing.

The workflow thread posts activities for processing by making an HTTP post to the Java Method Server or by issuing instructions to the Docbasic Method Server.

The workflow agent does not poll for additional activities if the number of activities assigned to the threads is more than five times the number of threads (5*n, where n is the number of threads). When the workflow thread task count drops below 5*n, the workflow agent queries the workflow queue and repeats the process for additional available activities. If there are no available activities the workflow agent sleeps for a period defined by the polling interval. This sleep interval is defined in `dm_server_config` and is called `wf_sleep_interval`.

Increasing workflow threads and adding Content Servers

Adding workflow threads increases system capacity to complete activities. Adding workflow threads also creates additional demand on the database and Content Server resources, which can affect TaskSpace performance or the performance of any other application using the same infrastructure (Increasing workflow threads on the TaskSpace Content Server, page 29).

Iteratively modify the number of system workflow threads to assess the impact on user response time, activity throughput, and system resource consumption. More workflow threads result in greater automatic activity throughput up to the point where system resource consumption degrades performance. Scale up slowly to understand when resource limitations begin to show (Content Server CPU and database CPU utilization). The following provides some guidelines:

- A single CPU Content Server host cannot process 10,000 activities per hour, regardless of how it is configured.
- Be cautious if CPU or memory utilization exceeds 80% for any tier in the system.
- Do not configure more than three threads per CPU core.

If throughput requirements exceed the capacity that a single Content Server can provide, add more Content Servers. Each Content Server instance (and associated workflow agent) nominally supports 15 concurrent workflow threads. Deploy one Content Server instance for every multiple of 15 concurrent workflow threads required by your solution. Avoid more than 25 workflow threads for any Content Server.

Note: If your throughput requirements exceed the capacity of the database, adding additional Content Servers does not help. To determine the capacity of your database, monitor your database CPU. If there is no idle CPU, your database hardware is probably undersized.

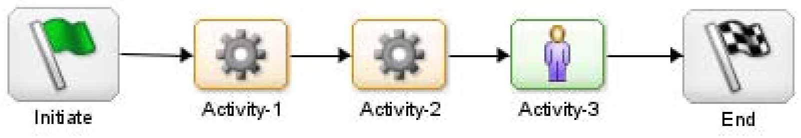
When adding Content Servers, balance the load of concurrent workflow threads across each new Content Server and initially configure each new Content Server with fewer than the nominal maximum number of concurrent workflow threads (15). For example, if you have 15 workflow threads on one Content Server and you introduce another Content Server with the same configuration, lower the number of threads to 10 each (20 threads total), then scale up the threads slowly to the required number.

Each new Content Server includes a workflow agent that can poll the database, which can slow down the database. Configuring the workflow agent (polling), page 29 provides more information on polling intervals and on-demand processing.

Maximizing throughput across all workflow instances

The Content Server workflow agent maximizes automated activity throughput by processing activities, from any active workflow, in the order in which they are written to the work queue (first in, first out). Sometimes, users do not see manual activities in their inbox until the automatic activities of many other workflows complete, even though there are only a couple automatic activities preceding the manual activity on which they act (Figure 7, page 29).

Figure 7. Simple workflow



Increasing workflow threads on the TaskSpace Content Server

For systems consisting of multiple Content Servers, dedicate one Content Server to TaskSpace and the rest to processing activities (Dedicating a Content Server to TaskSpace, page 29). If TaskSpace must use a Content Server that also processes workflow activities, incrementally add additional workflow threads while monitoring TaskSpace performance. Use the highest number of workflow threads where acceptable TaskSpace performance can still to be achieved.

When deciding between adding additional threads or Content Servers, choose additional Content Servers if the resources are available.

Dedicating a Content Server to TaskSpace

To remove Content Server resource contention between workflow threads and TaskSpace performance, provide dedicated Content Servers to handle the workflow threads and a different dedicated Content Server for TaskSpace. To prevent a Content Server from processing automatic tasks, make the following configuration setting in the Content Server `dm_server_config` object:

```
wf_agent_worker_threads=0
```

Configuring the workflow agent (polling)

The workflow agent can be configured for polling or on-demand processing. When configured for polling, the workflow agent periodically polls (queries) the database work queue for batches of tasks (up to 30 per workflow thread) to be processed. When no activities exist in the queue, the workflow agent 'sleeps' for a duration of time (seconds) set by the polling interval (the default polling interval is 5 seconds). When the polling interval expires, the workflow agent polls the database.

The workflow agent only sleeps when no activities exist in the queue to process. When activities exist, the polling interval has no effect because the workflow agent does not sleep.

When configured for on-demand processing (Configuring for on-demand processing, page 31), Content Server notifies the workflow agent when new activities complete and new work items get created in the work queue. The workflow agent then queries the database for the new work item and the workflow agent picks up the new work queue item for processing.

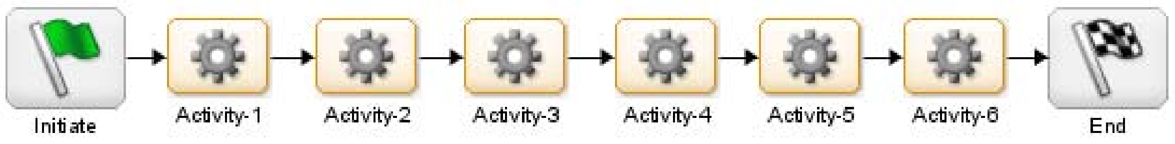
On-demand processing can result in many more queries to the database than when polling for batches of work items (especially for active work queues), which results in a common database performance problem. On-demand processing can also result in an unbalanced load across Content Severs as each Content Server only processes the automatic tasks its workflows generate. With polling, all Content Servers can process tasks generated by any of the other Content Servers.

Polling can also provide for load balancing, even if on-demand processing is enabled. For example, a Content Server that runs out of its own on-demand activities for the duration of the polling interval, polls the database for tasks in the common queue.

Increasing throughput for single or low volume workflows

In a single workflow (Figure 8, page 30), processing of a single automatic activity exhausts the backlog of activities in the workflow queue until the automatic activity can be completed and the next activity in the workflow enters the workflow queue. During this temporary lull, in which the queue does not contain any activities for processing, the workflow agent goes into sleep mode for the duration set by the polling interval.

Figure 8. Single workflow



If a workflow has 6 steps and the polling interval is set to 30 seconds, the workflow accumulates 150 seconds of sleep time before the workflow completes. The following illustrates the sequence of behaviors.

1. Complete workflow #1, Activity 1
2. Sleep 30 seconds
3. Complete workflow #1, Activity 2
4. Sleep 30 seconds
5. Complete workflow #1, Activity 3
6. Sleep 30 seconds
7. Complete workflow #1, Activity 4
8. Sleep 30 seconds
9. Complete workflow #1, Activity 5
10. Sleep 30 seconds
11. Complete workflow #1, Activity 6

Decreasing the polling interval improves throughput for similar scenarios with single workflows or low volume workflows.

Increasing polling intervals for multiple Content Servers

The workflow agent polls the database by issuing two queries to the database each time the polling interval expires. If your deployment comprises eight Content Servers, eight workflow agents (one for each Content Server) poll the database. A polling interval of 5 seconds for each of these 8 Content Servers results in 8 x (60/5) x 2= 192 queries a minute to the database or 11,520 queries per hour. In contrast, a 5-minute polling interval results in only 96 queries an hour.

To compensate for the increased load on the database that polling from multiple Content Servers creates, increase the polling intervals for each Content Server incrementally by the amount of the polling interval for one Content Server. For example, if the polling interval for one Content Server is 15 seconds, set the polling interval for two Content Servers at 30 seconds, three Content Servers at 45 seconds, and so on.

Configure the workflow agent for all (equally sized) Content Servers in a multi-server deployment the same way, except if one Content Server also services TaskSpace. If one Content Server also services TaskSpace, configure it differently to ensure sufficient levels of TaskSpace performance.

Configuring the polling interval

The following example uses IAPI to configure the polling interval for 30 seconds:

```
API> retrieve,c,dm_server_config
API> set,c,l,wf_sleep_interval
SET> 30
API> save,c,l
```

Note: The repository must be restarted for these configuration changes to take effect.

Configuring for on-demand processing

Content Server can be configured to process automatic workflow activities as soon as they are created. This configuration provides the lowest latency and is useful for test systems or when there are few users (low load) and few automatic activities (low throughput). Enable on-demand processing by setting the following environment variable for your operating system:

```
DM_NOTIFY_WF_AGENT_AUTO_TASK=T
```

With on-demand processing, Content Server immediately notifies the workflow agent upon creation of an automatic activity and does not wait for the polling interval. When using on-demand processing, set the polling interval high to enable Content Server and workflow agent to complete their transactions long before expiration of the polling interval. Setting the polling interval to 300 seconds generally provides the best throughput.

Note: Use polling for Content Servers with workflows launched more frequently than every 3 seconds.

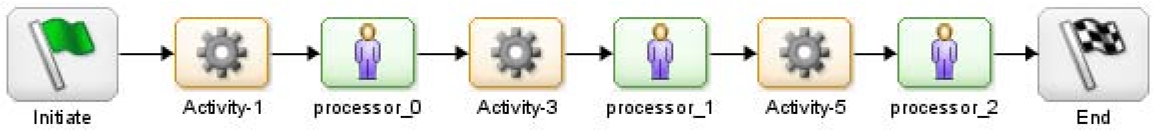
Avoiding manual bottlenecks

Throughput bottlenecks can result from an inability to process automatic or manual activities. Manual activities slow down throughput the most because they require human performers. Human performers process work items more slowly and their availability can be unpredictable. Workflows that designate specific individuals as performers of an activity can become stuck when that individual becomes unavailable for some reason (sick, vacation, and so on). To prevent manual bottlenecks resulting from specific individual performers:

- Assign manual activities to groups or queues, not individual users.
- Use timers for completion of manual activities and alert an administrator when a task has not been completed in the allotted time.
- Use automatic activities wherever possible. Most workflows have a completion rate ratio of 10 automatic activities to 1 manual activity.
- Develop a contingency plan to address a backlog if it does occur. Make sure that you can reassign tasks to other people, groups, or queues.

Figure 9, page 32 shows a workflow example in which a performer of a manual activity goes on vacation or there is an influx of workflows and the tasks build up.

Figure 9. Potential manual bottlenecks



When delays in expected workflow completion occur, run the following DQL command:

```
select distinct name, count(*) from dmi_queue_item group by name
```

```
processor_0  10
processor_1  24,000
processor_2  1
```

The result of this query indicates that processor_1 has 24,000 tasks in their queue before all workflows involving that performer can be completed. The result indicates that tasks are not being distributed evenly (load balanced) across all available processors, with processor_1 being the bottleneck.

Sample workflow agent configurations

This following provides sample workflow agent configurations.

On-demand processing — In this configuration, a single user or a few users log in to the system to demonstrate or verify something. They check their task list, and open and complete tasks. Workflow agent threads do not degrade end-user performance. The system polls once every 5 minutes. Content Server notifies the workflow agent upon creation of every automatic activity and the workflow agent picks up the job for processing.

```
polling interval=300
Set DM_NOTIFY_WF_AGENT_AUTO_TASK=T
Workflow agent threads =1
```

High user load, low automatic task throughput — This configuration optimizes application response time for end users when a high latency for the next manual task is acceptable. In this configuration, the workflow agent polls once every 2 minutes and hands the work off to a single workflow agent.

```
polling interval=120
Workflow agent threads =1
```

Low user load, high automatic task throughput — This configuration optimizes throughput at the expense of response time for TaskSpace users. If the priority is for throughput and degraded response time for TaskSpace users is not acceptable, consider deploying an additional Content Server.

```
polling interval=15
Workflow agent threads =15
```

High user load, high automatic task throughput — This configuration optimizes both throughput and response time for TaskSpace users.

```
polling interval=15
Several content servers with Workflow agent threads =15
```

Chapter 4

Designing the Application

This chapter provides guidelines for improving performance of TaskSpace forms and includes the following topics:

General design guidelines

As you design your application, keep in mind the following general guidelines:

- Avoid excessive subtyping in your object model. Each layer in your object model involves additional joins of database tables, which affects performance. Keep the inheritance hierarchy in your object model as flat as possible.
- Tune your queries for the highest *yield*, where yield is the number of times a query executes multiplied by query execution time. In some cases, achieving the highest yield involves de-normalizing a database table by pulling attributes into one or two physical tables.
- Avoid designing forms with functions that are not likely to be used often. The best performing applications enable users to complete their jobs with a minimum number of actions and choices.

- Only track essential events in the audit trail. Auditing can slow down response times and increase CPU usage on the Content Server and database.
- Partition the workload and avoid bottlenecks.

Preventing high load user actions

Individuals can engage in certain activities that put excessive demands on the system and affect overall system performance. For example, a case insensitive partial keyword search across several types can lock the database until the query can be completed. Running a resource-intensive job or report can also slow down system performance.

Design your application to prevent high load user scenarios from occurring. During development and testing, devise scenarios that can slow down system performance and design these scenarios out of your application.

Improving login speed

The user login operation takes 2-10 seconds to complete. The landing page that opens after a user logs in affects login time the most. For fast logins, set the landing page to the default page or to a blank search page.

Note: User preferences for the selected landing page negatively affect the performance improvement.

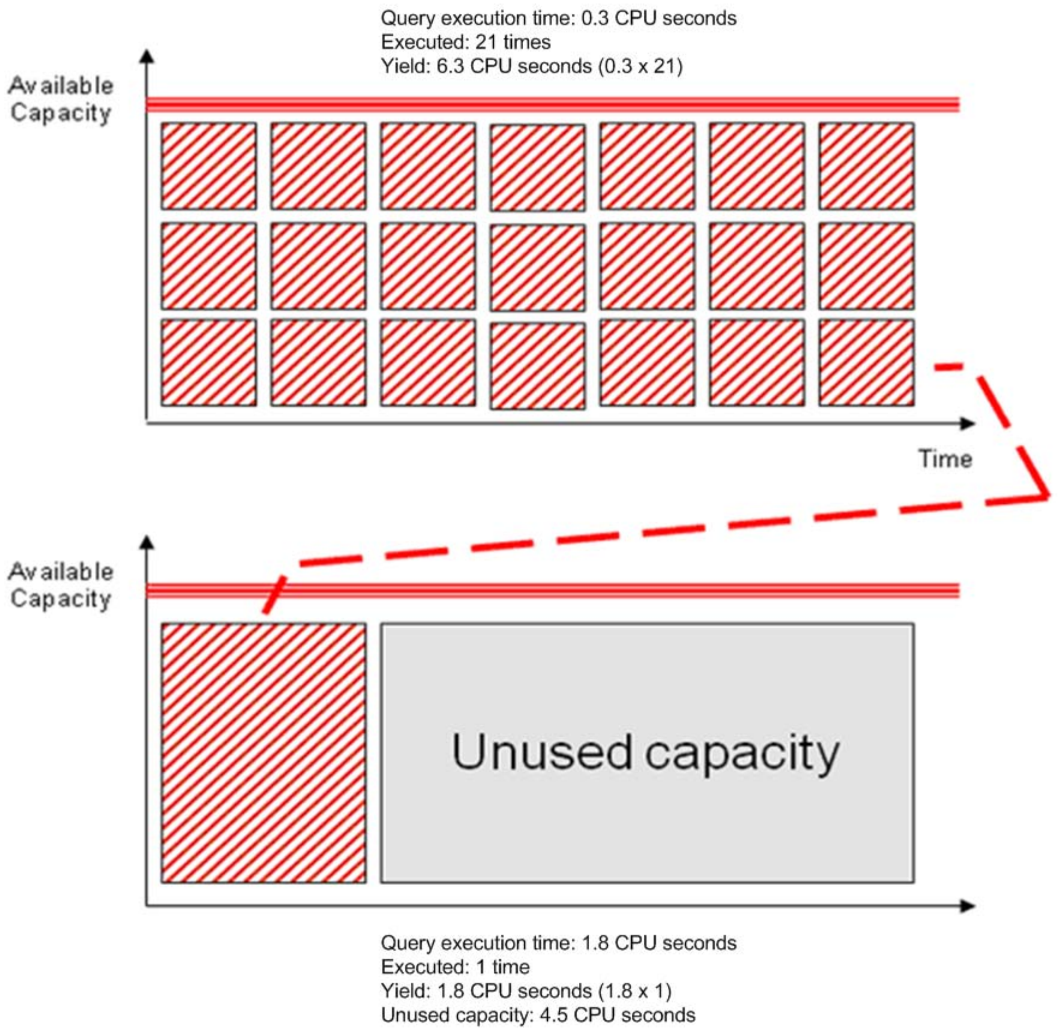
Maximizing query yield

Focus your query tuning effort on those queries providing the highest yield. The number of times a query executes multiplied by query execution time determines the yield.

Figure 10, page 37 illustrates the possible yield of two queries. The first (fast) query executes 21 times and each instance takes 0.3 seconds to execute. The second (slow) query executes one time and takes 1.8 seconds to execute. Because the fast query executes more frequently (21 times) than the slow query (1 time) a 10% improvement in the fast query execution time reclaims more CPU bandwidth (0.63 seconds) than a 10% improvement in the slow query execution time (0.18 seconds).

Even though there can be more latitude for improving slow queries, the frequency with which the query gets executed often results in a larger aggregate impact. A small percentage improvement to frequently executed fast queries can provide a better yield than a large improvement to infrequently executed slow queries.

Figure 10. Query yield


Query execution time: 0.3 CPU seconds
Executed: 21 times
Yield: 6.3 CPU seconds (0.3 x 21)
Available Capacity
Time
Available Capacity
Unused capacity
Query execution time: 1.8 CPU seconds
Executed: 1 time
Yield: 1.8 CPU seconds (1.8 x 1)
Unused capacity: 4.5 CPU seconds

Designing the process object model (using structured datatypes)

Process Builder uses process variables to represent different types of data that flow through your business process. These process variables are defined by a single attribute of a specific type (string, Boolean, date, and so on) or by a complex Structured Data Type (SDT) consisting of multiple attributes, each of a specific type.

Caution: Pay close attention to your object model design as your choices can greatly affect application performance and are difficult to change once your application is in production.

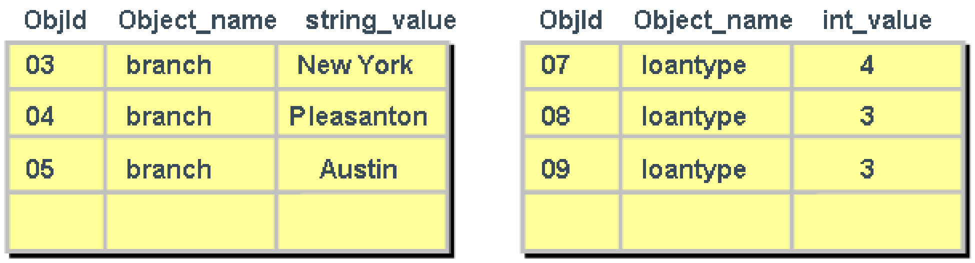
Note: Using simple (single attribute) process variables instead of SDTs has a pervasive negative affect on performance across various xCP functions, including search, task listing, and BAM reporting.

The database creates a separate table for each object type used, whether it is an SDT containing many attributes or a simple process variable containing a single attribute. When searching for attributes in different database tables, the database dynamically joins rows of database tables in order to retrieve pertinent data. Joining rows of database tables impacts performance. When using SDTs, a single database table contains the searchable attributes and avoids the performance impact of the join operation.

Tip: Use SDTs to consolidate sets of attributes to the minimal number of object types on which your business process discretely operates.

Figure 11, page 38 illustrates how the database models two simple process variables of different types (string and int), resulting in two different database tables. Any search against these two process variables employs multiple unions for each simple process variable, which slows down performance. In addition, the database cannot create a composite index, which would otherwise provide additional performance improvements (Creating composite indexes, page 38).

Figure 11. Separate data tables for different process variable types

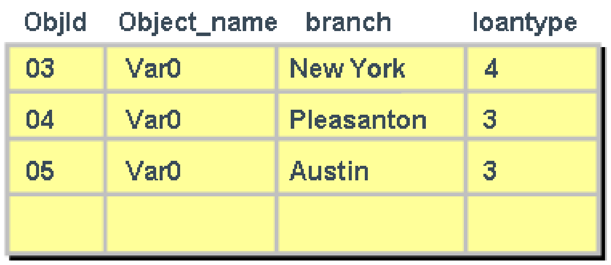


ObjId	Object_name	string_value
03	branch	New York
04	branch	Pleasanton
05	branch	Austin

ObjId	Object_name	int_value
07	loantype	4
08	loantype	3
09	loantype	3

Model the two process variables (branch and loantype) as attributes of a single SDT, in which case the database stores all metadata in a single database table (Figure 12, page 38). The database table can then be searched without the performance overhead of creating unions, as in Figure 11, page 38. In addition, you can create a composite index for SDTs, which provides additional database performance improvements (Creating composite indexes, page 38).

Figure 12. Single table representation for an SDT



ObjId	Object_name	branch	loantype
03	Var0	New York	4
04	Var0	Pleasanton	3
05	Var0	Austin	3

Creating composite indexes

xCP provides developers with the ability to create composite indexes for SDT attribute values. Creating a composite index can improve database performance, especially when performing complex (conditional) searches against a large database of different attributes in the same SDT.

Note: Composite indexes cannot be created across different types (simple process variables) or across single value and repeating value attributes in the same type.

Note: Have a database administrator determine which columns of a table to index. Creating too many indexes can cause update and insert statements to perform poorly.

See *EMC Documentum Content Server DQL Reference Manual* and *EMC Documentum Content Server Fundamentals* for information on using `make_index` to create a composite index.

Minimizing fetch operations

During operations like opening a task, xCP extracts process variable attribute values and displays the data on a form. For each process variable whose attribute displays on a form, xCP performs a fetch operation for the process variable and its parent object. Design forms that require the fewest number of fetch operations (involve the fewest number of process variables) as each fetch operation affects performance.

For example, when using simple process variables only, a form with 50 fields performs 100 discrete fetches. When using a single SDT to provide the 50 fields of information, the form performs only 2 discrete fetches, which results in far better performance.

Note: Forms that perform fewer than 60 fetches open in 2-3 seconds. Reducing the number of fetches to less than 60 does not result in significant performance improvement.

Note: Content Server 6.6 provides for caching of the parent object after initial fetching. As a result, the number of fetches (after the initial fetch) reduces by half (1 per process variable rather than 2 per process variable).

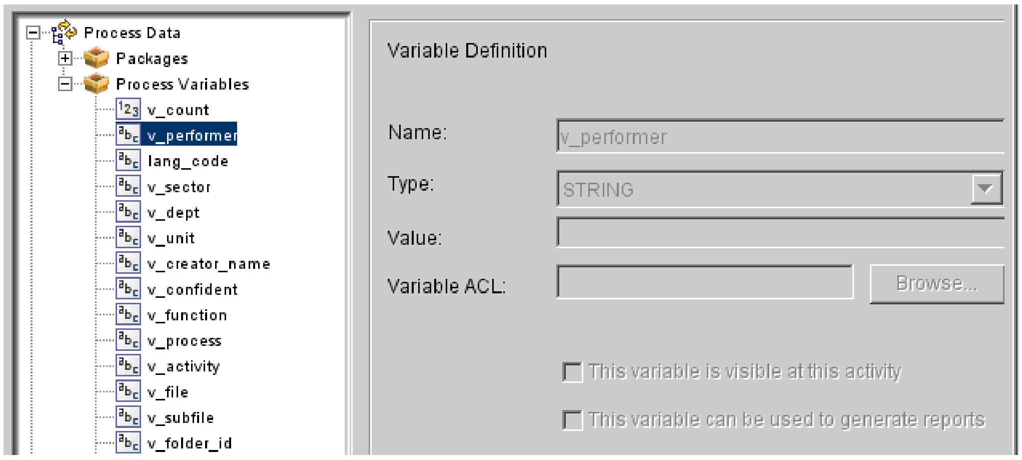
Note: Making a process variable invisible does not prevent an object fetch but it does result in a slight performance improvement (Hiding unused variables, page 39).

Analyzing process variable usage (measuring fetches), page 75 provides information on measuring the number of object fetches in your application.

Hiding unused variables

Process Builder provides the option to mark process variables as visible for manual activities. Making a process variable invisible improves performance by reducing processing overhead associated with the form. If no form fields use a process variable, hide the process variable by deselecting the checkbox in the process variable definition (Figure 13, page 40).

Figure 13. Hiding a process variable



Process Data
Packages
Process Variables
v_count
v_performer
lang_code
v_sector
v_dept
v_unit
v_creator_name
v_confident
v_function
v_process
v_activity
v_file
v_subfile
v_folder_id

Variable Definition

Name: v_performer

Type: STRING

Value:

Variable ACL:

☐ This variable is visible at this activity

☐ This variable can be used to generate reports

Converting simple process variables to SDTs

If your design makes extensive use of simple process variables and you can use SDTs instead, create new SDTs with attributes that map to the previously used simple process variables. Replace as many process variables as possible with SDT attributes. Performance improves even if you consolidate the simple process variables into several (instead of only one) SDTs. After creating the SDT, delete the simple process variables for which the SDT provides an attribute.

If your application is already in production and you cannot consolidate simple process variables into SDTs, hide those simple process variables that do not show up in any forms (Hiding unused variables, page 39) If there are many process variables that are mandatory on the form because they are mapped to a field and the application is in production and cannot be changed, create a workflow template you can use while migrating off the old template.

Chapter 5, Designing Reports provides information on the potential performance impact of updating the BAM database with an SDT definition.

Minimizing form size

Form size determines form rendering performance more than any other single factor. To maximize form rendering performance, consider the following:

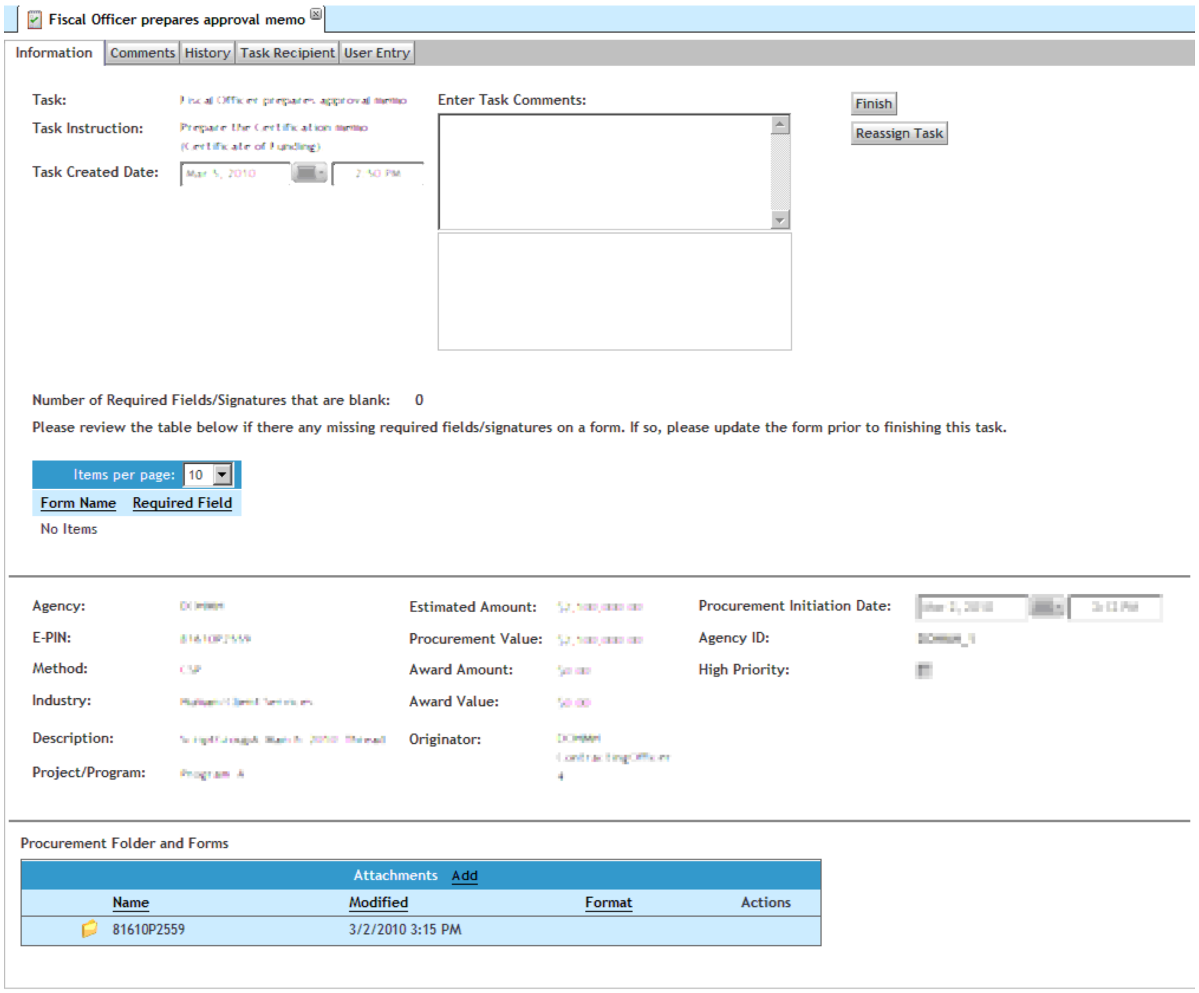
- Only use packages when necessary.
- Adaptor-based data increases form footprint and affects response time (Using adaptors, page 52).
- Use virtual documents or embedded folders, instead of tabs, to contain the forms or documents needed for processing.

Form size impacts system sizing. A typical JVM can support anywhere between 40-200 users, depending on the size of the forms involved in day to day business activities. The larger the forms, the fewer users the JVM can support.

Figure 14, page 41 provides an example of a large form with the following characteristics:

- 46 packages (3 visible)
- 20 fields mapping
- two embedded folders
- four additional tabs
- 30 adaptors.

Figure 14. Sample form



Fiscal Officer prepares approval memo

Task:	Fiscal Officer prepares approval memo	Enter Task Comments:	
Task Instruction:	Prepare the Certification memo (Certificate of Funding)		
Task Created Date:	Mar 5, 2010	2:50 PM	

Number of Required Fields/Signatures that are blank: 0

Please review the table below if there any missing required fields/signatures on a form. If so, please update the form prior to finishing this task.

Items per page: 10

Form Name	Required Field
No Items	

Agency:	[illegible]	Estimated Amount:	[illegible]	Procurement Initiation Date:	[illegible]
E-PIN:	[illegible]	Procurement Value:	[illegible]	Agency ID:	[illegible]
Method:	[illegible]	Award Amount:	[illegible]	High Priority:	
Industry:	[illegible]	Award Value:	[illegible]		
Description:	[illegible]	Originator:	[illegible]		
Project/Program:	Program A				

Procurement Folder and Forms

Attachments Add

Name	Modified	Format	Actions
81610P2559	3/2/2010 3:15 PM		

Figure 14, page 41 represents a form that takes up 10 MB of heap space. In this case, a 1024 MB heap can only support about 80 users. The following breaks down the form components making up the 10 MB form size:

- Packages: 43 packages marked as visible add 2 MB per form instance in the JVM. For example, 200 users opening up this form consume 400 MB in the JVM. To minimize this impact, mark only the packages you need as visible. Set the rest to invisible.
- Forms Template: The DOM object created internally for this type of form composition consumes roughly 2 MB per instance.

- Text Fields: For a form with 100 text fields, the template and layout file sizes are 34 KB and 52 KB, respectively. During runtime, the corresponding forms processor size is 1 MB.
- Adaptors: Processor requirements spike for forms using adaptors that generate large amounts of data. One adaptor can generate data that translates into 300 XML nodes. This results in the form processor generating about 10,000 new objects to represent the read/write, hide/show, and valid/invalid properties for the XML nodes.

Designing search forms

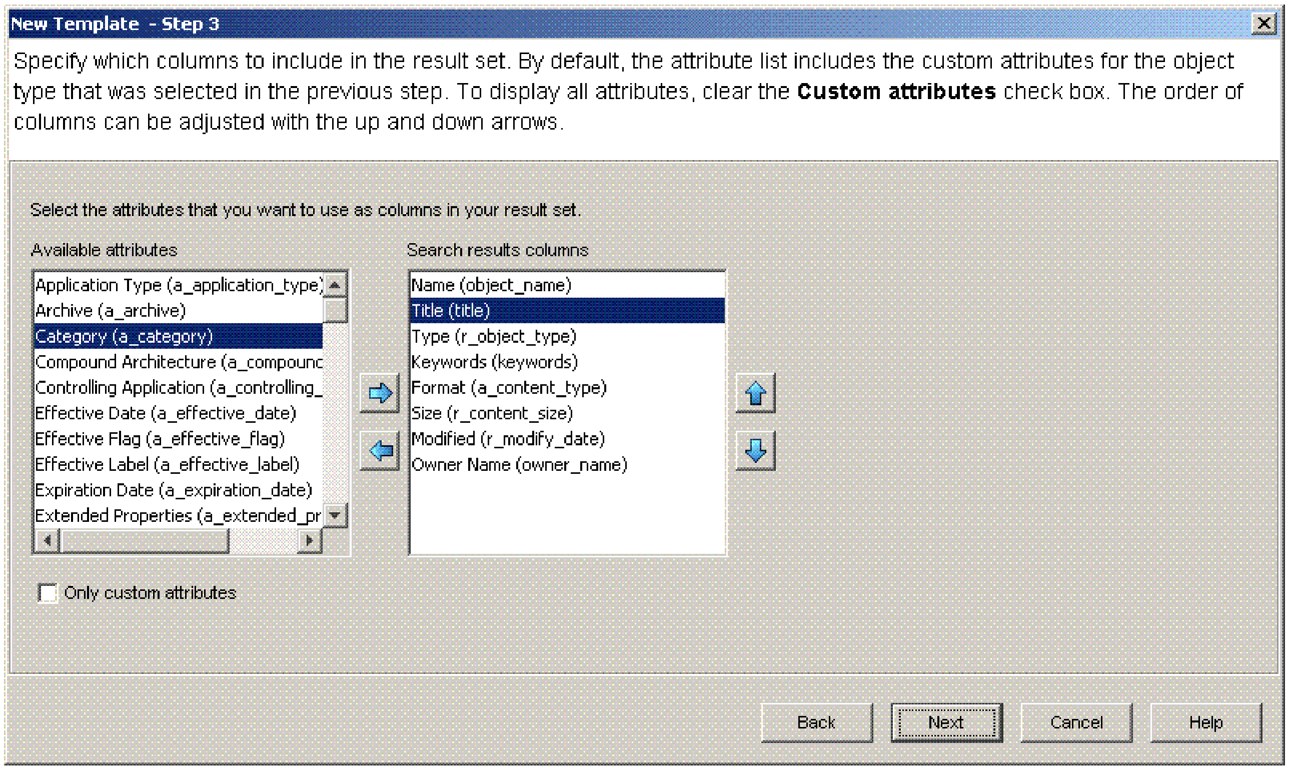
TaskSpace component templates (created in Forms Builder) define the set of objects, stored in the repository, that get searched for (queried) in TaskSpace forms. To maximize system performance, avoid unnecessary functions (querying) in your templates and consolidate your queries so that the fewest number of queries provide required data for a TaskSpace form. Design your object model to use SDTs instead of simple process variables (Designing the process object model (using structured datatypes), page 37). Only use search criteria that is necessary for users to complete their jobs. The following sections provide specific performance tips related to search-related functions configured for TaskSpace forms.

- Using searchable attributes, page 42
- Using search criteria, page 43
- Using search restrictions, page 45
- Using advanced searches, page 45

Using searchable attributes

The search template, task list template, folder contents template, and process instance list template include a screen, like the one shown in Figure 15, page 43 for the search template, to specify attributes to be searched and the columns in which to display the results of the search.

Figure 15. Mapping object type attributes to columns in which search results are displayed



New Template - Step 3

Specify which columns to include in the result set. By default, the attribute list includes the custom attributes for the object type that was selected in the previous step. To display all attributes, clear the **Custom attributes** check box. The order of columns can be adjusted with the up and down arrows.

Select the attributes that you want to use as columns in your result set.

Available attributes	Search results columns
Application Type (a_application_type)	Name (object_name)
Archive (a_archive)	Title (title)
Category (a_category)	Type (r_object_type)
Compound Architecture (a_compounc	Keywords (keywords)
Controlling Application (a_controlling_	Format (a_content_type)
Effective Date (a_effective_date)	Size (r_content_size)
Effective Flag (a_effective_flag)	Modified (r_modify_date)
Effective Label (a_effective_label)	Owner Name (owner_name)
Expiration Date (a_expiration_date)	
Extended Properties (a_extended_pr	

☐ Only custom attributes

Each searchable attribute and column in which the search result is displayed adds load to the system. Design search forms with as few searchable attributes and results columns as possible.

Wherever possible, use single-value attributes instead of repeating value attributes. TaskSpace combines all single value attributes into a single query, but issues an additional query for each repeating attribute. Each additional query affects performance.

Include the following attributes in your search form. Mark the attributes invisible unless they must be visible for business purposes.

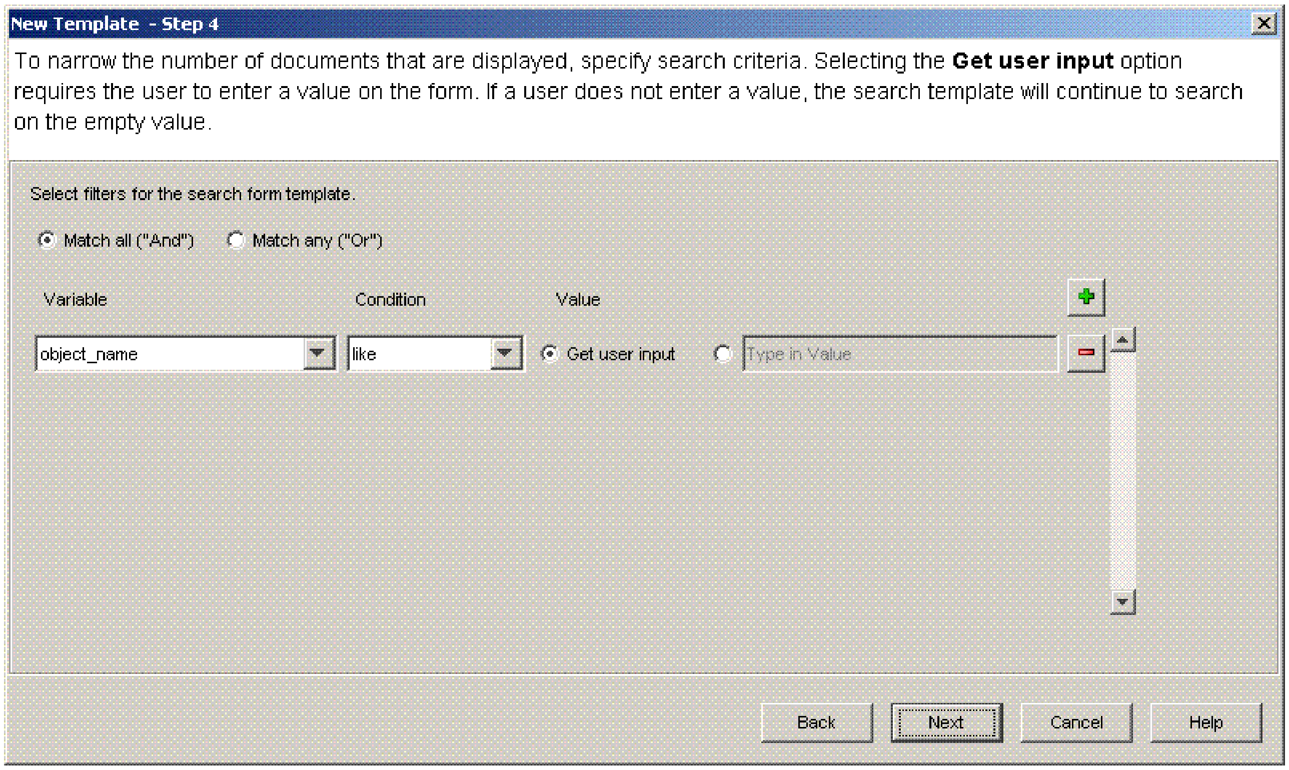
- `r_lock_owner`
- `a_content_type`
- `r_content_size`
- `r_object_type`
- `r_object_id`

If these attributes are not present in the search query of the form, TaskSpace performs a full fetch of the object to retrieve the attributes, which affects performance.

Using search criteria

The search template, task list template, folder contents template, and process instance list template include a screen, like the one shown in Figure 16, page 44 for the search template, to specify search criteria for each item selected for display in a results column.

Figure 16. Specifying search criteria


New Template - Step 4

To narrow the number of documents that are displayed, specify search criteria. Selecting the **Get user input** option requires the user to enter a value on the form. If a user does not enter a value, the search template will continue to search on the empty value.

Select filters for the search form template.

[✓ Match all ("And")] Match any ("Or")

Variable	Condition	Value
object_name	like	[✓ Get user input] Type in Value

The condition you select defines the syntax of a corresponding search query built by Forms Builder. The following table describes available conditions and indicates the resulting syntax for an example query:

Table 2. Search criteria conditions

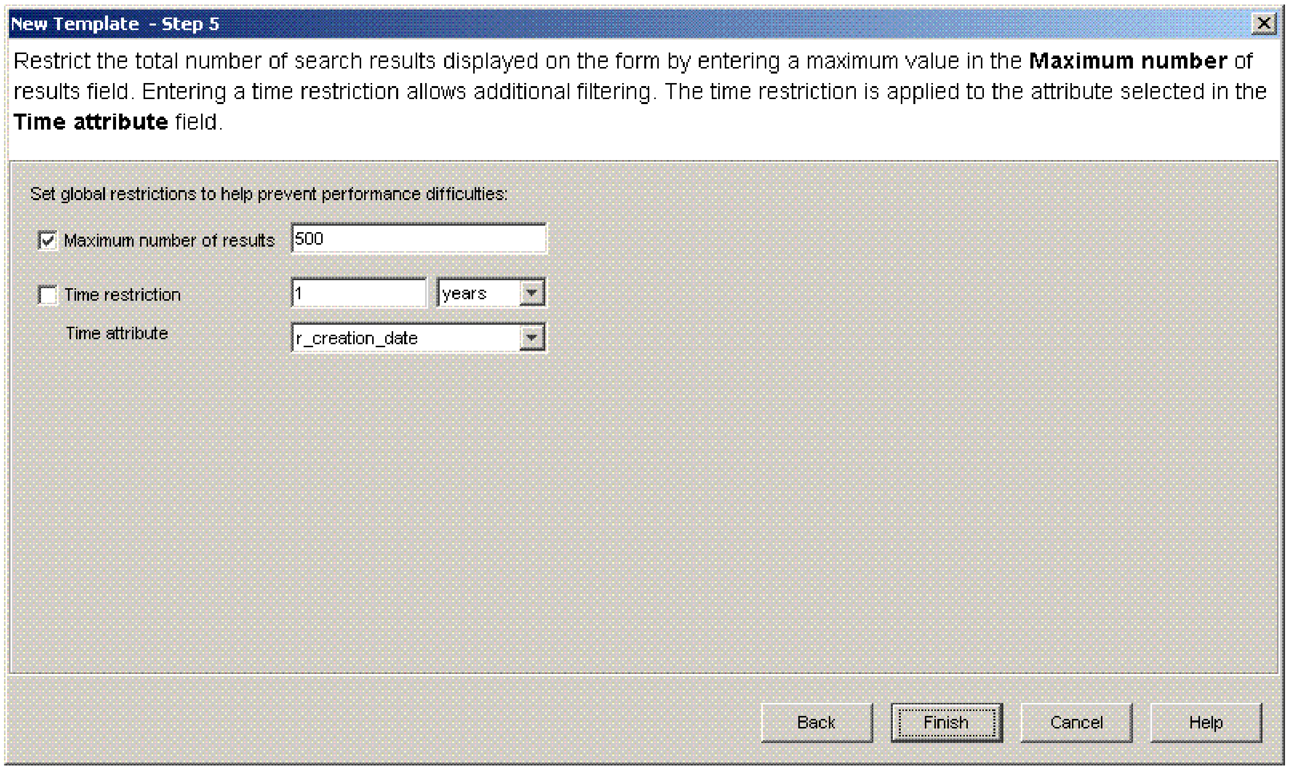
Condition	Description	Example
Equals	Returns items that match the specified value exactly.	object_name = '$object_name$'
Not equals	Returns items that do not match the specified value exactly.	object_name <> '$object_name$'
Like	Returns items that match a pattern with a wildcard.	object_name LIKE '%$object_name$%'
Not like	Returns items that do not match a pattern with a wildcard.	object_name NOT LIKE '$object_name$'
Starts with	Returns items in which the first part of the item matches the specified value.	object_name LIKE '$object_name$%'
Ends with	Returns items in which the last part of the item matches the specified value.	object_name LIKE '%$object_name$'

Exact searches scale much better than wildcard searches. Wildcard searches involve scanning of a full-text index or database table which affects performance, especially with larger repositories containing many objects. Avoid using the Like, Not like, Starts with, and Ends with conditions.

Using search restrictions

The search template includes a screen (Figure 17, page 45) to set restrictions on the results of a search.

Figure 17. Configuring search restrictions



Restrict the total number of search results displayed on the form by entering a maximum value in the **Maximum number of results** field. Entering a time restriction allows additional filtering. The time restriction is applied to the attribute selected in the **Time attribute** field.

Set global restrictions to help prevent performance difficulties:

☑ Maximum number of results	500
☐ Time restriction	1 years
Time attribute	r_creation_date

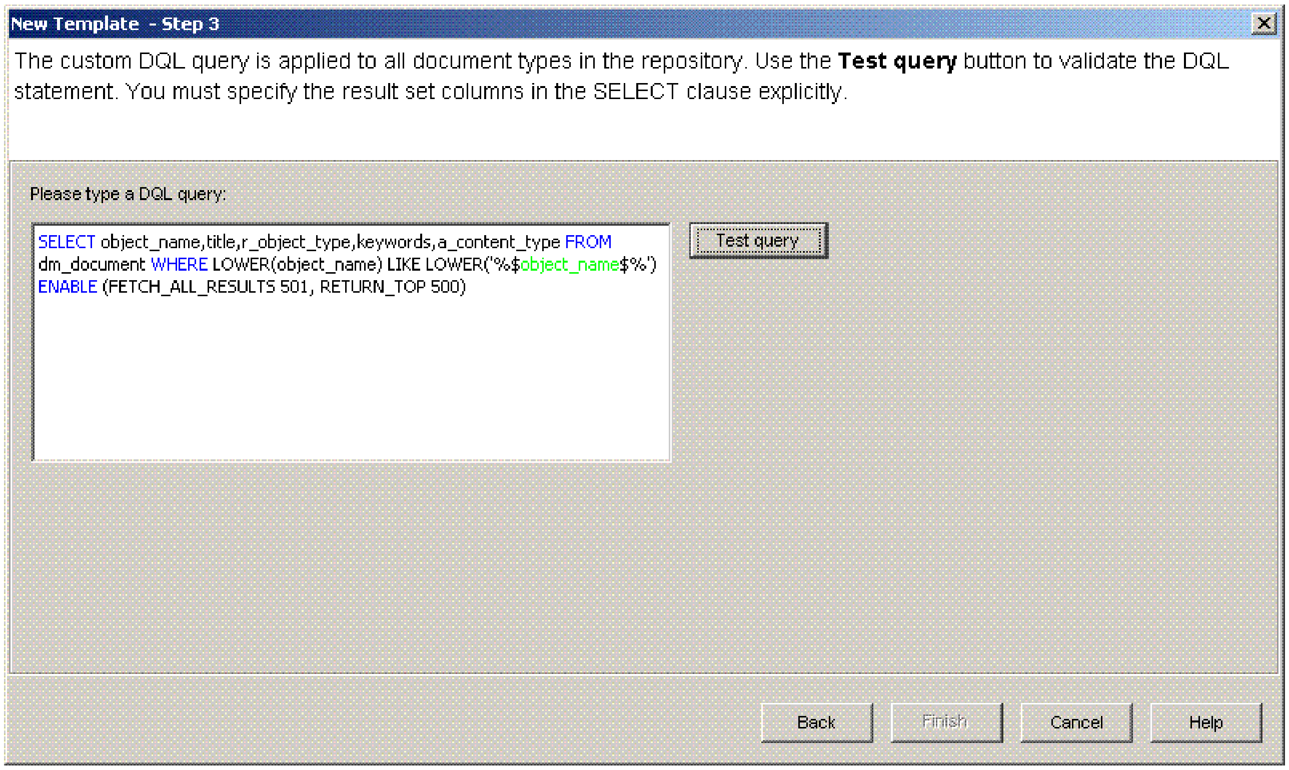
Set the search restrictions appropriately to return results of use only. Failure to specify a restriction on the number of results can have a significant impact on performance, especially when thousands of objects match the search criteria.

See Restricting advanced search results, page 46 for information on restricting results for advanced queries.

Using advanced searches

The search template includes a screen (Figure 18, page 46) to create a search query using DQL. More information on DQL is available in the *Documentum Content Server DQL Reference Manual*.

Figure 18. Interface for manually defining search queries



The custom DQL query is applied to all document types in the repository. Use the **Test query** button to validate the DQL statement. You must specify the result set columns in the SELECT clause explicitly.

Please type a DQL query:

```
SELECT object_name,title,r_object_type,keywords,a_content_type FROM
dm_document WHERE LOWER(object_name) LIKE LOWER('%$object_name$%')
ENABLE (FETCH_ALL_RESULTS 501, RETURN_TOP 500)
```

Note: Wildcards (**`SELECT *`**) are not supported in manual queries.

Follow the same performance-related guidelines in your advanced (manual) queries that are followed for simple queries, including minimizing the number of attributes searched and setting appropriate results restrictions.

Restricting advanced search results

Avoid designing search templates that do not restrict result sets. In the following example:

```
SELECT r_object_id, object_name, commodity_code, commodity_description
FROM e2e_commodity WHERE commodity_code LIKE '%$SEARCH_CODE$%'
```

the query provides placeholder variable (`%$SEARCH_CODE$%`) which the end user specifies at run time. If the query returns 10,000 results, all 10,000 results get returned to the TaskSpace application server after a single user click. The large number of results can result in poor performance for the user performing the search and the overall TaskSpace application.

To improve single user and overall TaskSpace performance, restrict query result sets by adding `ENABLE (RETURN_TOP n)` to the DQL query.

```
SELECT r_object_id, object_name, commodity_code, commodity_description
FROM e2e_commodity WHERE commodity_code LIKE '%$SEARCH_CODE$%' ENABLE (RETURN_TOP 501)
```

The `ENABLE (RETURN_TOP 501)` restricts the query results to the first 501 rows that match the search criteria, even though there are many more possible matches.

Note: Adding criteria to your query results in fewer items returned and less performance impact.

Note: Content Server, version 6.6, uses range queries that limit results sets to 100.

Designing task lists (work queues)

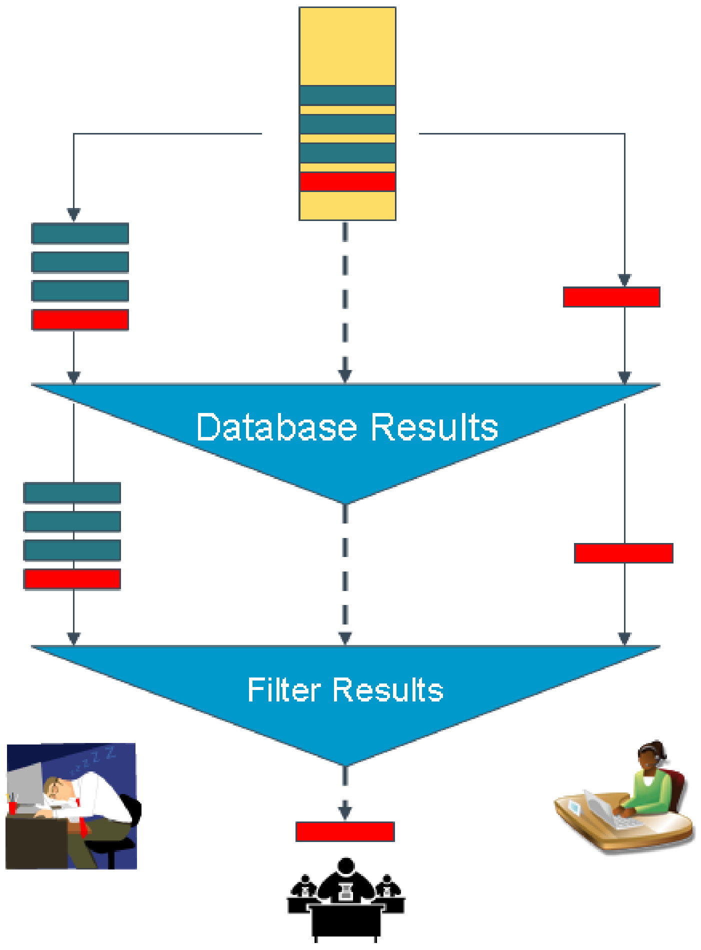
In TaskSpace, you create work queues to specify tasks performed by any one of multiple individuals. The work queues can then be specified as the performer of an activity defined in Process Builder. You define work queues by specifying the members (individuals, groups, or roles), policies (priorities), and work assignment matching filters associated with the work queue.

TaskSpace populates task lists of individual users by querying work queue repository objects when the associated work queue gets invoked as the performer of a process activity. TaskSpace applies work assignment matching filters to the query results, returned from the repository, to refine the list of tasks included in a user's (processor's) task list.

Avoid creating work queues with large 'unselective' membership. Unselective work queues return large result sets to TaskSpace which then undergo row by row processing through work assignment matching filters before a list of tasks for a user inbox can be updated. With unselective work queue memberships, row by row processing through TaskSpace filters compounds the performance impact that results from the database returning a large result set of potential tasks for which a user is unqualified (left branch of Figure 19, page 48.

Designing work queues with 'selective' membership results in small result sets returned by the database, better database performance, and fewer rows for processing through work assignment matching filters (right branch of Figure 19, page 48). In general, assign work queue members that are qualified to work on most tasks and the highest priority tasks in the queue. Use the work assignment matching filters to fine-tune task assignments.

Figure 19. Processing for unselective and selective work queues



Designing skill-set matching

When using work queues with skill sets, design the skill set levels to apply evenly across tasks in the work queue. For example, if there are 2 skill levels defined in the skill set for a work queue with 100 tasks, design the skill levels so they result in approximately 50 tasks being uniformly distributed. Unbalanced skill set filtering can create bottlenecks and sluggish Get Next Task performance in the work queues of those individuals with the less discriminatory skill values.

Skill set matching works like a precondition (Avoiding unnecessary preconditions, page 50) in that TaskSpace sequentially evaluates each row in the task list against user competency. Furthermore, skill set matching requires evaluation of the entire task list. For large task lists (10,000 items, for example), evaluating an entire task list can have a significant performance impact.

Establish a baseline performance metric for your form before adding skill sets. Assess the performance impact of adding skill set matching against the baseline to determine whether the value offered by skill set processing is worth the performance cost.

Rendering task lists

Poor performance for task list rendering (list of eligible tasks take a long time after user clicks the tasks tab) can occur because the task list query returns many tasks, additional row by row processing occurs on task list query results, or both. To maximize performance of task list rendering:

- Design your work queues and skill sets properly (Designing task lists (work queues), page 47.
- Minimize the use of custom preconditions (Avoiding unnecessary preconditions, page 50).
- Consider partitioning large work queues into several smaller work queues (Filtering a task list and partitioning a work queue, page 49).
- Constrain the query results for large result sets (Constraining query results for large result sets, page 49).

Note: If a task list includes pseudo attributes (specifically package names), another query is issued to get the object names of the package documents associated with each task.

To measure task list rendering performance, collect a DFC trace on the TaskSpace application server for a single user. The DFC trace shows all the queries issued when user clicks the tasks tab. Analyze the task list query to determine the performance issue with the query (Chapter 7, Measuring Performance). Hot fixes and future releases (6.6, for example) eliminate some additional queries.

Filtering a task list and partitioning a work queue

System capacity can be exceeded when working with large work queues (approximately 50,000 tasks) during peak loads. In such cases, partition the work queue into multiple queues that satisfy different criteria, or design forms that filter displayed tasks by date range (for example: today, last seven days, and so on). Both these approaches limit the number of results returned to a task list.

Troubleshooting the get next task function

The get next task function can be implemented by clicking the Get Next Task button or by setting a preference in TaskSpace so that the next task automatically displays after completion of a task.

When the get next task function executes automatically, the get next task stored procedure executes first, followed by rendering of the task form. Performance issues with the get next task function can result from problems with the task form, the get next task procedure, or both.

To isolate whether there is a problem with the form, collect a DFC trace while manually clicking on the task list to open a task, which does not involve the get next task procedure. If rendering the form manually performs well, focus on the procedure. If the form performs poorly, focus on the form itself.

Constraining query results for large result sets

Poor performance for task list rendering can result when a work queue contains thousands of tasks and the task list query does not limit (constrain) the number of results returned from the database.

For example, if the initial task list page looks for tasks in a queue containing 10,000 tasks, unless the task list query imposes a limit on the number of results, the database returns all 10,000 results to Content Server and the application server for task list rendering. The performance impact of this scenario exposes itself through the following symptoms:

- The initial task list form takes a long time to render when a user clicks the "Tasks" tab in TaskSpace.
- The initial task list displays the tasks, but sorting on a column takes a long time to display the results.

View the collected DQL query for `ENABLE (RETURN_TOP n) hint` where n is the number of fetched rows. Lack of the `ENABLE (RETURN_TOP n) hint` indicates the following:

- The task list form contains a default sort criteria on a process variable (either primitive or SDT attribute) and this process variable is not used to filter results (not used in the filter criteria). Remove the default sort on this process variable from the task list form.
- The task list form displays multiple process variables (either primitive or SDT) and filtering occurs on a different process variable than sorting. The task list service cannot limit the results from the database and returns all results to the TaskSpace application server, where the application server does an in-memory sort. Carefully design the task list form to prevent certain columns from being sortable.

Avoiding unnecessary preconditions

TaskSpace can be customized using the Documentum Web Development Kit (WDK). One of the most common TaskSpace customizations affecting performance involves the use of preconditions.

TaskSpace forms that list rows of items (Tasklist, Search, and Folder View, for example), evaluate preconditions on result sets returned from a database query before displaying each row in the list. The preconditions evaluate attributes of listed objects to enable or disable actions that the user can perform on an object (edit or view only, for example). While preconditions provide for rich functionality, they can have a significant impact on performance.

TaskSpace processes all preconditions for each object in a list before displaying information about that object as a row on a form. As a result, preconditions exact a multiplicative performance impact on rendering a complete list in a form. For example, if there are 10 items in a list, each with 20 applicable preconditions, TaskSpace executes 200 preconditions before rendering the list. A task list that takes 10 seconds to render with preconditions can take 3 seconds to render without preconditions.

Establish a baseline performance metric for your form before adding preconditions. Individually assess the performance impact of each precondition against the baseline to determine whether the value offered by the precondition is worth the performance cost.

Note: Many preconditions inspect object attributes such as "object type", "content type", "content size", "lock owner", and so on. If these attributes are not present in the search query of the form, TaskSpace performs a full fetch of the object to retrieve the attributes, which adds an additional performance impact.

Changing the default date attribute for sorting a task list

Task list queries that use filtering and sorting against large database tables (many users and work items) perform poorly. By default, TaskSpace stores the data, against which filtering and sorting queries get executed, in different database tables. The performance impact can be due to the database spending resources on evaluating the sort instead of using an index (indexes do not span multiple tables).

Collect the database execution plan and note the sort query performance against the default queue item `date_sent`. If it performs poorly, use Forms Builder to modify the task list form to sort on the work item creation date, which is in the same data table as the default filter criteria. Validate that this change makes a substantive improvement by rerunning the database execution plan.

Using task history (audit trail)

Task histories provide a list of actions that were performed on items in a task list before the user received the task. TaskSpace queries the audit trail database table for task history metadata. The audit trail database table can often grow to be large. As a result, the queries for task histories can have a significant performance impact.

The following provides a DQL example used to populate a task history:

```
select workflow_id from dm_audittrail where event_source = 'workflow' and
event_name = 'dm_wf_invoke_subprocess' and id_5 = '4d99f81680014546'
```

which is then translated into this SQL:

```
select all dm_audittrail.workflow_id from dm_audittrail_sp  dm_audittrail
where ((dm_audittrail.event_source=N'workflow') and
(dm_audittrail.event_name=N'dm_wf_invoke_subprocess') and
(dm_audittrail.id_5=N'4d99f81680014546'))
```

The following index was applied to the table in order for the query to perform:

```
dm_audittrail_s(id_5,event_name,event_source)
```

The audit trail query could grow to 2 million rows, where it would have a major impact on performance. As a general guideline:

- Avoid using task histories unless they are essential.
- If you use task histories, maintain the audit trail tables by purging or archiving database table rows.
- Expand the `dm_audittrail` database table to a couple million rows to see how it impacts performance.

Embedding documents (document viewing)

Task forms containing embedded document (for viewing) typically add 2-3 seconds in total rendering time. When designed incorrectly, rendering time can become much longer (more than 10 seconds).

Document Image Services (DIS), in combination with Accelerated Content Services (ACS) and Branch Office Caching Services (BOCS), enable page serving for certain document formats with particular document viewers. See the *Document Image Services Deployment Guide* and *TaskSpace Deployment*

Guide for supported format and viewer combinations. For 6.5 SP2, DIS supports page serving for PDF and TIFF document formats with both Daeja ViewOne Pro and IGC Brava! document viewers. The following can cause long rendering time:

- The application server downloads the full document to the client instead of single pages. Large documents exhibit a more pronounced impact. Configure ACS for page serving (Turning on page serving, page 66).
- For Brava viewers, every request from the viewer goes to the Brava license viewer. Network latency with the Brava license viewer can result is poor performance.
- First time viewing takes more time. Browser caching improves performance after the first time view.

Checking ACS operation, page 79 provides information on measuring the performance of ACS operations.

Using adaptors

Form templates specify how to handle data associated with the form, such as initial data values, instructions for validating data a user enters, and storing the data when the form is submitted. Adaptors help complete a form by providing data for certain fields. For example, an adaptor typically populates a dropdown box with a list of countries, certain departments, or employees. Usually this information is not hard coded into the dropdown (although it can be), and is brought over through a query within the xCP system or from outside the system.

Designing adaptors

Forms open slowly when they contain too many adaptors, poorly written adaptors, or when there are problems with an adaptors external data source (network latency).

- Defer execution of the adaptor to some other event than form initialization, for example, when the user selects their inbox, or selects a tab or control. Deferring adaptor execution is especially important when an adaptor provides dependent data, like state/city, as this information requires multiple queries to the adaptor.
- Hard code values that do not change often, like country names, as they do not require much maintenance.
- Design forms for single use cases and verify the necessity for adaptor execution in each use case.
- Avoid integrating with external systems – calls to external systems are synchronous and dependent on the performance of the external system. If possible, implement an asynchronous solution that mitigates getting real-time data across a network from a third-party system or database.

Measuring adaptor performance, page 78 provides information on measuring adaptor performance.

Managing group memberships

The number of groups tends to increase with the number of objects in an application. When users belong to more than 250 groups, a threshold is reached and the DQL changes to accommodate the number of groups they are in, which results in degraded performance.

The following DQL statement gets a count from the dm_process object:

```
select count(r_object_id) as c from dm_process (all) where r_object_id in
('4b0011ec8005c985','4b0011ec800af4fd','4b0011ec8000431a','4b0011ec80004319')
and any sd_element_type = '080011ec8007d38b' and any sd_element_type = '080011ec800042d9'
```

When the user belongs to 249 groups, the above DQL takes 0.032 seconds. When a user belongs to 251 groups, the above DQL takes 12.688 seconds (almost 400 times slower). The performance impact of belonging to 250 or more groups affects most TaskSpace operations involving user object requests.

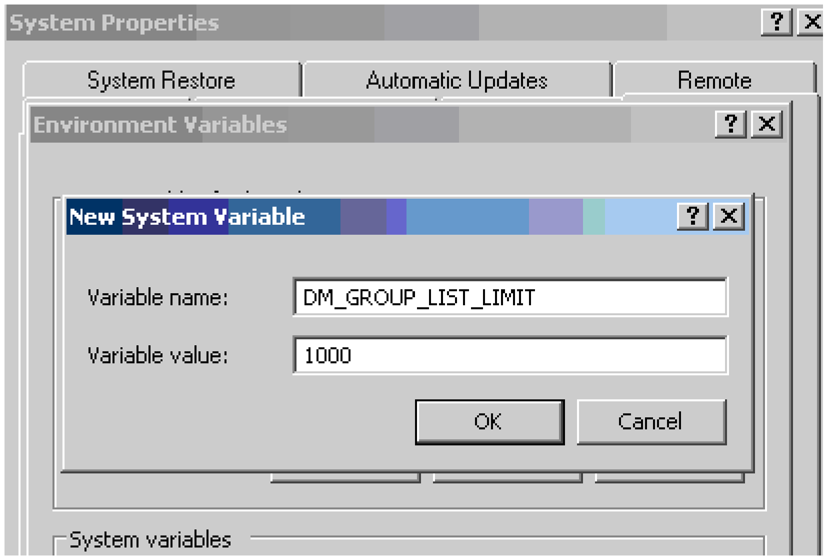
In addition to the 250 group membership threshold performance affect, every session the user creates caches the groups for which the user is a member. When many users belonging to many groups use the system, Content Server becomes consumed with caching groups for every new session, becomes unable to service other requests, and the application can shut down.

Working around the 250 group threshold

The 250 group membership threshold can be overridden by setting the group limit environment variable to something higher than 250, before hitting the 250 group threshold. Setting the group limit parameter maintains the behavior of the DQL for less than 250 groups, until the specified group limit is hit.

For example, using `DM_GROUP_LIST_LIMIT=1,000`, the query behavior stays the same until the user has over 999 groups. Set the group limit as an operating system parameter on the Content Server and restart Content Server for the setting takes effect. Figure 20, page 54 shows how the setting looks in a Windows environment.

Figure 20. Setting group limits



Variable name: DM_GROUP_LIST_LIMIT

Variable value: 1000

Designing Reports

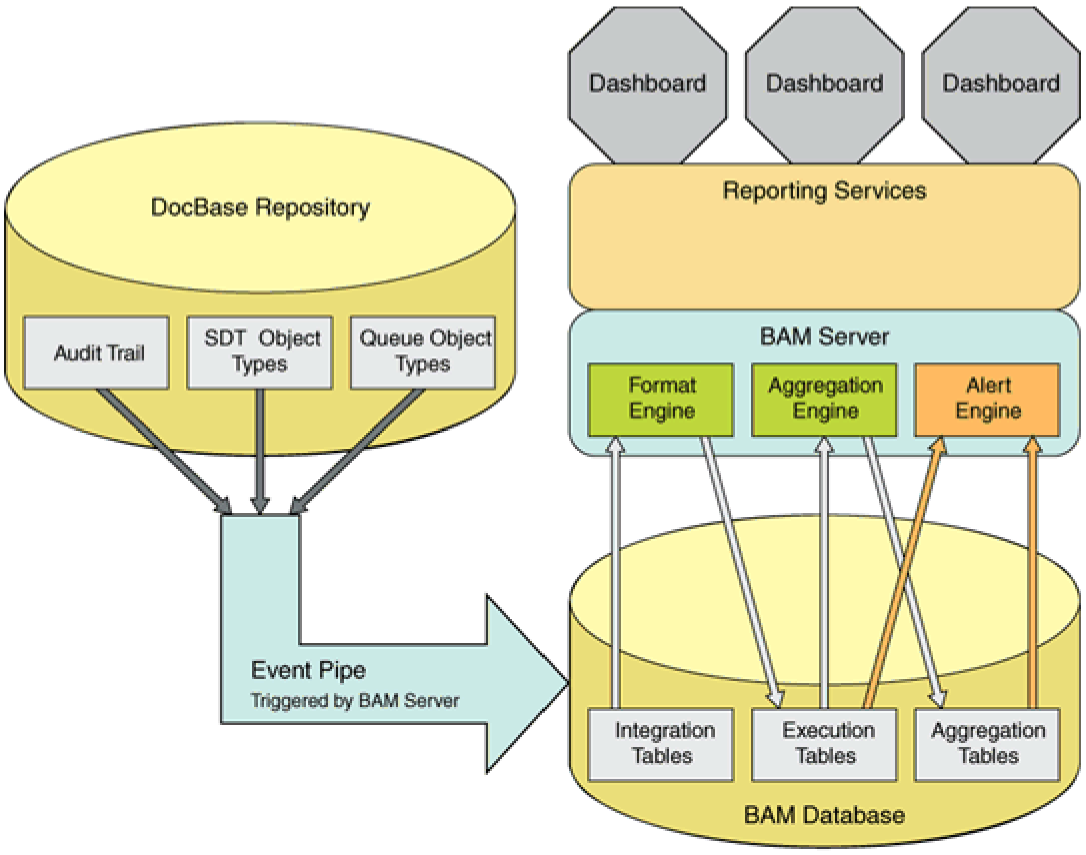
This chapter provides guidelines for improving performance of BAM reports and includes the following topics:

Understanding BAM reporting

By default, BAM monitors process and queue data (process and activity durations, information about performers, queue data, and started/finished date information) in the audit trail database table. Designers use Process Builder to configure additional data (activity package data and process variable data) to be written to the audit trail database table. Designers use TaskSpace to configure BAM to monitor the activity package data and process variable data in the audit trail database table.

During runtime, the Content Server process engine writes audited activity package data and process variable data to the audit trail database table upon completion of the containing activity (Figure 21, page 56).

Figure 21. BAM architecture



The BAM server controls operation of the event pipe and resulting transfer of data from the audit trail database table to the BAM database integration table. Each row of the integration table contains data about one process instance (activities, performers, queues, and business data). The format engine extracts information from the integration table and maps it into different execution tables for the different types of process instance data (activities, performers, queues, and business data).

The aggregation engine aggregates process instance data from the execution tables into groupings, for which one or more arithmetic functions apply, and stores the groupings in an aggregation table. The alert engine scans rows in the execution and aggregation databases and evaluates them for alert conditions defined in Process Reporting Services (PRS). The alert engine adds process instances (rows), that satisfy the alert condition, to the BAM database alert instance table.

Planning and testing

BAM reporting can be performance sensitive. To avoid performance issues, observe the following guidelines during your planning:

- Plan for reporting at the beginning of the project.
- Conduct performance testing of your BAM reports in a test environment that simulates your production environment regarding the following:
  - a fully loaded BAM database
  - anticipated volume of processes, activities, and audited business data
  - high frequency report viewing
  - multiple concurrent users.

  Benchmark throughput and response time for your test environment under worst case scenarios.
- Size the BAM database to meet or exceed expected data throughput. Database requirements can be sized using the *BAM Dashboard Sizing Calculator* spreadsheet, available in the BAM Database Sizing Calculator folder of the bam-product-supplemental_files.zip file.
- Consider the size of the result set comprising the report. For reports with more than a few hundred records, use Crystal Reports.
- Only audit and monitor process variables and packages that are reported on.
- Consider the number of entities queried when generating the report. If your result set is a few hundred records, use more entities (up to 20). If your result set is thousand of records, limit the number of entities to no more than five. The more columns selected, the fewer report entities used.

Reporting on intra-activity events

Content Server does not write activity data to the audit trail database table until an activity completes, which can delay reporting on events happening within an activity, especially when your process design consolidates multiple actions in the context of a single activity. To report on intra-activity processes, divide the activity into multiple, more granular, activities in your process design.

Note: Creating more activities can have an adverse performance impact on workflow throughput (Minimizing and consolidating activities, page 27).

Synchronizing the BAM database

BAM generates reports from BAM database tables. When the BAM database tables do not regularly synchronize with the audit trail database, BAM report data can become stale. This section addresses issues related to BAM database and audit trail database table synchronization. The following highlights some of these potential causes:

- BAM server is not sized correctly. Events can accumulate in the audit trail faster than the BAM server can process them (Sizing the system, page 22).
- Network has performance problems.
- Gap filling is consuming system resources (Using the gap filler, page 58).
- BAM latency settings are too low (Configuring data transfer latency, page 58).
- BAM server is down or monitoring has been deactivated in TaskSpace (Understanding server clock synchronization, page 59).
- Business data model is being updated (Updating business data (SDTs and package objects), page 60).

Using the gap filler

Content Server continuously writes audited data to the audit trail database table regardless of whether the BAM server monitors the data. When the BAM server goes offline, data transfer to the BAM database does not occur and an information gap results between the BAM database and the audit trail database.

To fill the information gap resulting from an offline BAM server, the gap filler feature can be invoked when the BAM server comes back online. The gap filler feature instructs the BAM server to assimilate the backlog of data, back to the time configured by the recovery period.

The gap filler takes time to assimilate the backlog of data. During this time, the gap filler uses a substantial amount of system resources, which can affect system performance. For scenarios requiring frequent gap filling, conduct the gap filling during system down time or off peak hours.

Configuring data transfer latency

The BAM server queries the audit trail database table every 5 seconds for events in the audit trail database with time stamps 30 seconds earlier. The BAM server retrieves bundles of monitored event data, occurring within the 5-second intervals, and writes the data to the integration database. For example, a query issued at 1:20:00 retrieves events with time stamps from 1:19:25-1:19:30.

The default 30 second offset (data transfer latency) between the time of the BAM query and the time stamp of audited events queried for, compensates for the latency between when an audited event occurs and when Content Server writes event data to the audit trail database table. The 30-second data transfer latency works for conditions where Content Server writes audited information to the audit trail database table within a few seconds of activity completion.

Note: xCP versions before 6.6 use a default data transfer latency of zero seconds, instead of 30 seconds, which can easily result in missed report data. Configure the default data transfer latency to a value appropriate for your system requirements and capability.

In high load environments (more than 100 concurrent workflow events), the latency between activity completion and writing of audited data to the audit trail database can exceed 30 seconds, resulting in missed report data. For example, if a query issued at 1:20:00 retrieves events with time stamps from 1:19:25-1:19:30, but those events do not arrive in the audit trail database until 1:19:45 (a 45-second latency), the BAM query misses them. Subsequent BAM queries look for events with different time stamps, so BAM never reports on the late arriving event data.

The 30-second default data transfer latency can be changed to accommodate longer latencies in production environments. Use Documentum application system performance metrics to derive system latencies so that you can configure the data transfer latency appropriately. Use TaskSpace to configure the data transfer latency.

Note: Longer configured data transfer latencies, while ensuring completeness of monitored data, result in longer delays for updated report data.

Increasing the BAM server step size

Every 5 seconds the BAM server runs the event pipe job that extracts data from the audit trail database and inserts it into the BAM database. This five-second interval is the step size, and can be adjusted in high volume environments.

Although there is no absolute definition, 500 events or more per minute is considered high volume. Increasing the step size enhances performance in two ways:

- Although the BAM server runs the event pipe job less frequently, the job runs longer and more efficiently.
- Since the job is running less frequently, the size of each transaction is larger than for a step size of 5 seconds. Larger transaction sizes also make the BAM server operate more efficiently.

The BAM server step size can be adjusted to any time period by substituting a value for "60" in the following query. The following procedure increases the BAM server step size to 1 minute.

1. Shut down the BAM server.
2. Run the following query on the BAM database:

   ```
   I_BAM_SERVER_CONFIG SET SCHEDULEPERIOD = 60 WHERE SERVERNAME = 'DCTMServer'
   ```
3. Restart the BAM server.

Understanding server clock synchronization

The Content Server, BAM application server, and BAM database server clocks must be synchronized to within 5 seconds, with an outer limit of 30 seconds. In clustered environments the clocks between each Content Server within the cluster must be synchronized. Lack of synchronization can result is loss of BAM data.

Note: The BAM server clock synchronizes itself with the Content Server clock by reading the timestamps of monitored data in the audit trail database.

Updating business data (SDTs and package objects)

SDT and package object attributes can be aggregated and used as base entities in BAM reports. In contrast, simple process variables cannot be aggregated and must be nested under a Process Execution report entity. SDT and package object attributes can be updated in the BAM database when there are changes. The update operation creates or modifies report entities and BAM database tables, and can have a performance impact during the update operation.

Designing high performance reports

Design dashboards and reports to display necessary information only. The more data displayed in BAM reports, the greater the demand on system resources.

Note: For high volume environments with greater than a few hundred records, use Crystal Reports. For low volumes of process data consisting of a few hundred aggregated rows, use simple (PRS) reports.

Defining report entities and filters

The entities specified in your reports define the queries run against the BAM database. The query results comprise the report data. The larger the result set returned by the report queries, the greater the utilization of system resources and potential impact to performance. Use the following guidelines to minimize the size of your query results:

- Configure BAM to restrict the number of results returned in a report (Modifying the number of records returned in a results set, page 61).
- Use filters to restrict the result set to the rows matching the filter condition. For example, when the "Last Day" filter is used in a report, only a fraction of the data is queried and returned. Another example similar in its benefit is the user/role wildcard filter, which only displays the data related to the user viewing the report.

  Note: Avoid creating complicated filter expressions. Every condition in the filter expression adds complexity to the query, which slows down query performance.

  Note: Custom filters for SDT and activity package data can require creation of additional database indexes (Indexing the database, page 86).
- Configure initial/default filters to display reports with a predefined filter.
- Use mandatory filters to prevent reports with large volumes of data from displaying in the dashboard.
- Configure reports to drill into other reports. Each report applies its own filters, which reduce the volume of data processed by any one report.

  Note: Dashboard (runtime) filters are executed in-memory, not in the database query, and are less effective than PRS (design time) filters.
- Minimize the number of columns reported on for each entity.

- Use database materialized views for reports with complex queries that require joins between tables or calculations of the data. The database calculates views more effectively and quicker than the report engine.
- Use Crystal Reports to create data analysis reports, such as pivoting, that require repetitive filtering of data. For these types of reports, export report data to a .csv file and use tools such as Microsoft Excel to generate pivot reports.
- Design performance sensitive reports in a separate dashboard so they do not affect response time for the entire dashboard.

Modifying the number of records returned in a results set

The number of records returned for a report can be increased or decreased by modifying the `maxRows` parameter in the `jdbc-config.xml` configuration file. Controlling the number of records retrieved is especially important for reporting, since large results sets can negatively affect system performance and lead to an out-of-memory error. The BAM server returns 1,000 rows by default.

Note: When set to zero, the JDBC driver controls the maximum number of returned records.

Working across large data sets

To filter or sort across columns in different tables with large data sets, create a table that contains columns from all the sorted or filtered tables. Create an index on that table. Performance improves at the expense of storage and update overhead.

Using aggregation

Report summaries can be calculated from execution tables each time a report loads into the dashboard or calculated in advance and saved in the aggregation table. Report summaries generated from the aggregation table use less memory and provide shorter response time than report summaries generated from execution tables, especially for execution tables containing thousands of rows of data. Aggregation tables can be created for standard aggregation entities, custom entities, and business data (SDTs and package objects).

Aggregating high volume data tables

Creating summaries of execution data in aggregation tables requires system resources and can affect performance in high volume systems processing thousands of instances per day. In such cases, aggregate report data every 5 minutes to strike a balance between providing real-time report data, minimizing the system impact of generating reports directly from the execution tables, and minimizing the system impact of running aggregation reports against large sets of report data.

Using 5-minute aggregation enables near real-time dashboard reports by collapsing report data, within each 5-minute time interval, into a single record. BAM then performs calculations on fewer table rows than without aggregation. Also, since BAM aggregates report data every 5 minutes, the aggregation process does not have to deal with as much data as when a 30-minute aggregation interval is used. Provide additional performance improvement by filtering the aggregation report to include current data only since the last time the aggregation engine ran.

Refreshing the dashboard

BAM dashboards can be configured to refresh after a certain interval of time. Refreshing a dashboard consumes system resources. Choose a refresh interval that meets the business requirement to see updated data and considers the frequency with which the data changes. For low volumes of long running processes, a refresh interval of every few days can suffice. For have high volumes of rapidly changing processes, use more frequent refresh intervals.

Chapter 6

Configuring System Components

The database, Content Servers, and application servers can be configured to improve scalability and performance. The following sections provide information for configuring these servers:

Configuring the TaskSpace application

The *EMC Documentum TaskSpace Deployment Guide* and *EMC Documentum xCelerated Composition Platform Best Practices Guide* provide information on configuring the TaskSpace application server. The following topics provide information for configuring the TaskSpace application.

Disabling drag and drop

Depending on form complexity, drag and drop can incur up to 5 seconds of performance overhead. To disable drag and drop, set the `<dragdrop>` tag in the <*TaskSpace home*>`\taskspace\app.xml` file to a value of `false`.

```
<dragdrop>
    <!-- drag and drop is turned on by default -->
       <enabled>false</enabled>
</dragdrop>
```

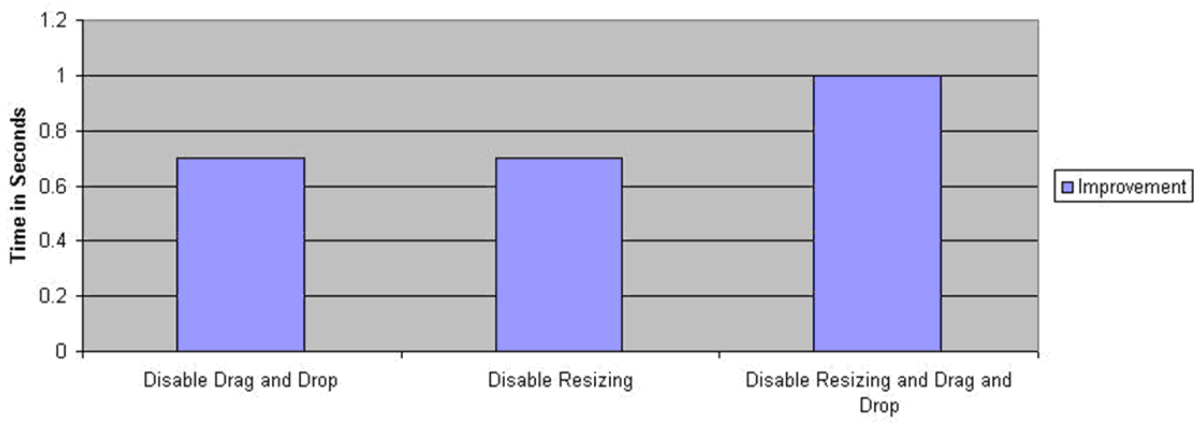
Disabling data-grid resizing

Resizing of data grids, like drag and drop, can incur a significant amount of performance overhead. If you disable drag and drop and data-grid resizing, you can achieve an 18% improvement in response time for task list rendering (Figure 22, page 64).

Disable data-grid resizing by adding the `<columnresize>` parameter in the <*TaskSpace home*>`\taskspace\app.xml` file and setting the value to `false`.

```
…
  <desktopui>
       <datagrid>
           …
           <columnresize>false</columnresize>
       </datagrid>
  </desktopui>
…
```

Figure 22. Seconds saved by disabling drag and drop, and data grid resizing



Increasing cache time

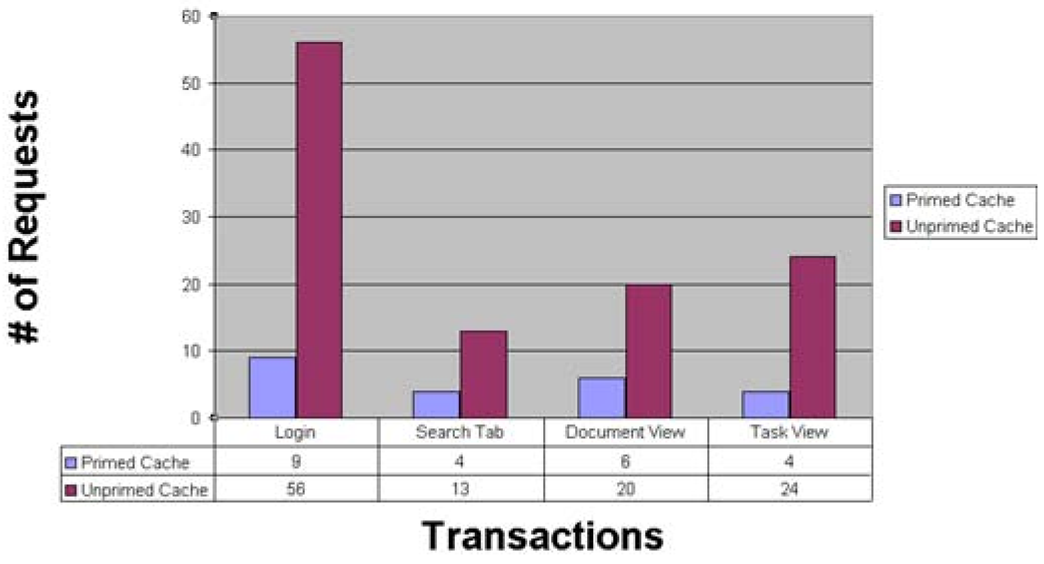
TaskSpace client-side caching provides significant single-click performance improvement by reducing the overall number (Figure 23, page 65) and size (Figure 24, page 65) of requests associated with common transactions.

Set the value of the `ClientCacheControl` filter in the <*TaskSpace home*>`\WEB-INF\web.xml` file to a high number as follows:

```
<filter>
      <filter-name>ClientCacheControl</filter-name>
      <filter-class>com.documentum.web.servlet.ResponseHeaderControlFilter</filter-class>
      <init-param>
         <param-name>Cache-Control</param-name>
         <param-value>max-age=604800</param-value>
      </init-param>
```

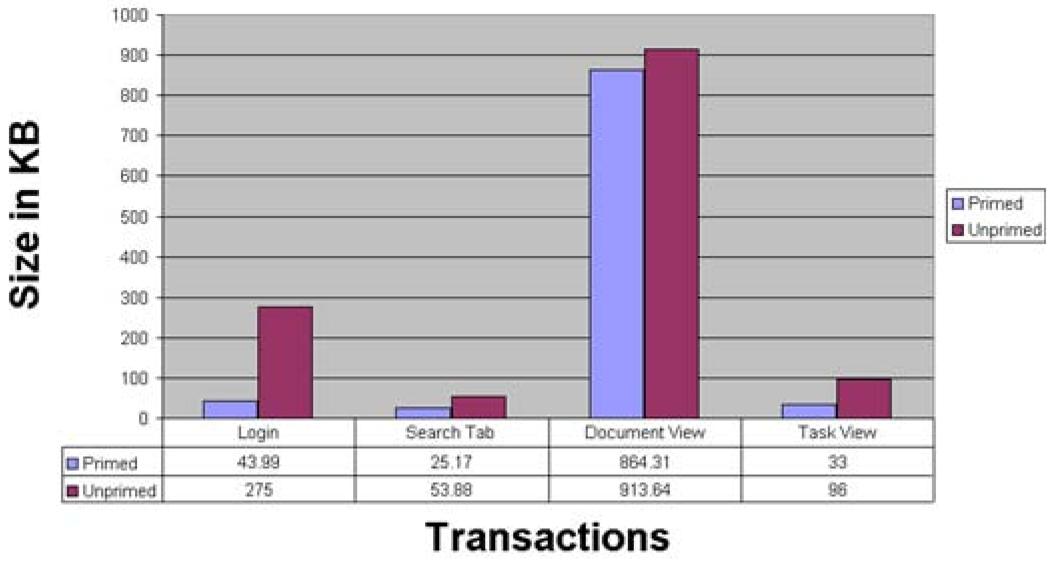
Configure the browser not to clear the cache when the browser closes.

Figure 23. Caching impact for number of requests



	Login	Search Tab	Document View	Task View
■ Primed Cache	9	4	6	4
■ Unprimed Cache	56	13	20	24

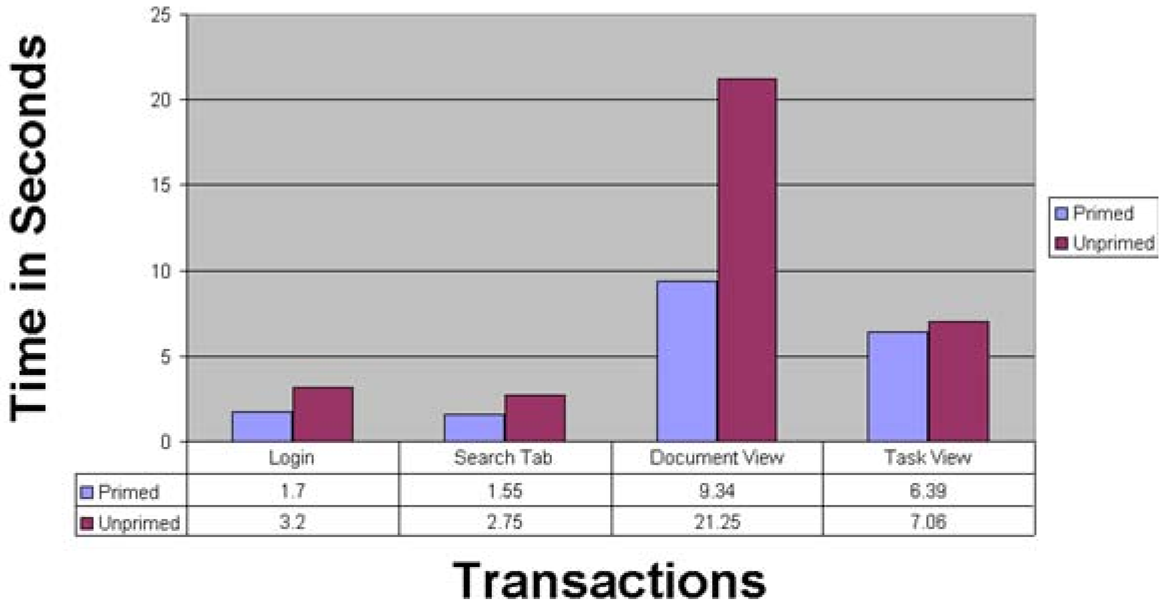
Figure 24. Caching impact for size of request



	Login	Search Tab	Document View	Task View
■ Primed	43.99	25.17	864.31	33
■ Unprimed	275	53.88	913.64	96

Caching also reduces response time associated with common transactions (Figure 25, page 66).

Figure 25. Caching impact for response time



	Login	Search Tab	Document View	Task View
Primed	1.7	1.55	9.34	6.39
Unprimed	3.2	2.75	21.25	7.06

Turning off Java annotations

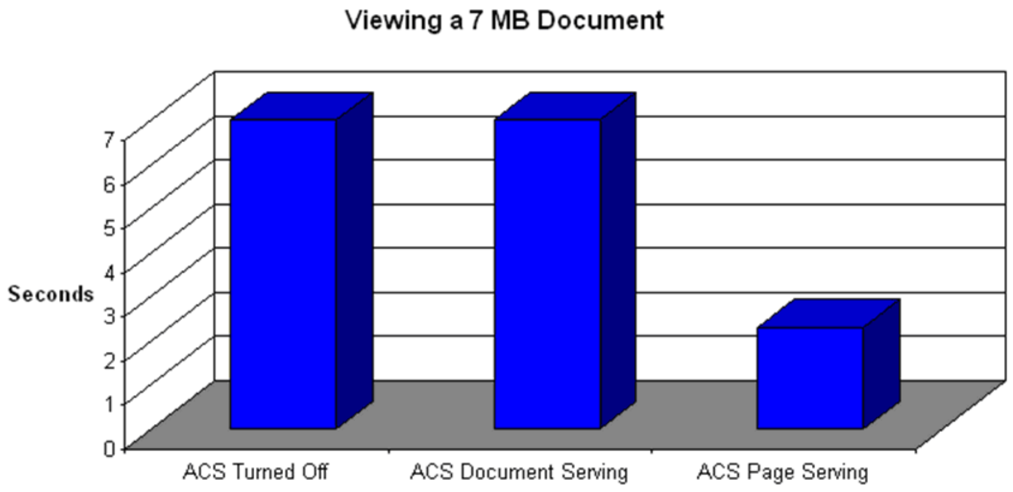
TaskSpace does not support the Java annotation feature. The Java annotation feature can have a significant TaskSpace performance impact (approximately 10%) on Tomcat 6.x and later application servers. For Tomcat 6.x and later, turn off the annotation feature by setting the `metadata-complete` parameter in the <*TaskSpace home*>`\WEB-INF\web.xml` file to `true`.

Note: You do not need to turn off the annotation feature for Tomcat 5.x and earlier.

Turning on page serving

The ACS page serving configuration provides a substantial performance improvement (Figure 26, page 67) for document viewing.

Figure 26. Document viewing with ACS page serving



Set the following ACS configuration data in the `{clientserver}\webapps\taskspace\wdk\config\imaging\ ivf_config.xml` configuration file:

- Set Content Server ACS ON/OFF.
- Set Page Serving Level or Document Serving Level.
- Set Page Serving Level Threshold (in bytes).

Note: Not all file formats can use ACS.

Configuring the BAM application server

The *EMC Documentum Business Activity Monitor Installation Guide* provides general information on configuring the BAM application server.

Configuring Content Server

The *EMC Documentum Content Server Installation Guide* provides general information on configuring Content Server.

Turning off debugging

Process Engine debugging runs in the Java Method Server (JMS), a JBoss application server. Process Engine debugging results in additional I/O and CPU resource consumption. By default, debugging is on.

To turn off debugging, copy and paste the following to the `<!-- Preserve messages in a local file -->` section of the *$DOCUMENTUM*`\jboss4.2.0\server\DctmServer_MethodServer\conf\jboss-log4j.xml` file:

```
<appender name="FILE" class="org.jboss.logging.appender.DailyRollingFileAppender">
   <errorHandler class="org.jboss.logging.util.OnlyOnceErrorHandler"/>
   <param name="File" value="${jboss.server.log.dir}/server.log"/>
   <param name="Threshold" value="INFO"/>
   <param name="Append" value="false"/>

   <!-- Rollover at midnight each day -->
   <param name="DatePattern" value="'.'yyyy-MM-dd"/>

   <!-- Rollover at the top of each hour
   <param name="DatePattern" value="'.'yyyy-MM-dd-HH"/>
   -->

   <layout class="org.apache.log4j.PatternLayout">
      <!-- The default pattern: Date Priority [Category] Message\n -->
      <param name="ConversionPattern" value="%d %-5p [%c] %m%n"/>

      <!-- The full pattern: Date MS Priority [Category] (Thread:NDC) Message\n
      <param name="ConversionPattern" value="%d %-5r %-5p [%c] (%t:%x) %m%n"/>
       -->
   </layout>
</appender>
```

Disabling email notifications

To turn off mail notification, add `MAIL_NOTIFICATION=F` to the startup section of the *$DOCUMENTUM*`\dba\config\documentum\server.ini` file. Restart the repository for the changes to take effect.

Configuring the BAM database

The *EMC Documentum Business Activity Monitor Installation Guide* provides general information on configuring the BAM database.

Chapter 7

Measuring Performance

This chapter provides guidelines for measuring performance and includes the following topics:

Measuring latency and throughput

Latency and throughput provide the two key Documentum system performance metrics. Latency defines the minimum time required to get any kind of response, even one that requires no processing time. Throughput defines the number of transactions the system can process per unit of time, 100 TaskSpace searches per minute, for example.

Single user and multi-user (load) testing comprise the approach for testing xCP performance. Single user testing measures latency by capturing metrics for business transactions at the client tier (HTTP request service times) or at the application server tier (Content Server RPC service times) on a quiet system. Single user testing involves collecting and analyzing detailed trace files for DFC and/or Oracle, and is performed before multi-user testing.

Multi-user testing builds on the single user testing by capturing service times for the same business transactions used in single user testing, executed by multiple concurrent users. Multi-user testing measures throughput by capturing "average" and "95th percentile" response times for a given business transactions under a given user load while measuring coarse system statistics (CPU utilization, network utilization, disk I/O, and so on). Multi-user testing response times can never be any better than single user response times.

Single user testing can be used to measure the user experience from various locations (laptop via VPN or remote facility, for example). User experience testing can account for network latency, browser Javascript processing overhead, or other client machine activities such as virus scan, disk encryption, or stateful firewalls. User experience testing use HTTP proxy tools (Charles, for example) to capture HTTP requests and service times on the client browser machine. Load testing tools like WinRunner (unlike LoadRunner) capture and measure the user experience but have to be carefully architected to account for desired testing experience (network bandwidth, Javascript processing, virus scan, disk encryption, stateful firewalls, and so on).

Troubleshooting high latency

Consider the case where a business transaction searches a large repository and TaskSpace is configured to display 100 items per page. There can be so much data matching the search criteria that it cannot all be cached in memory either by Oracle or by the storage array, in which case the query can take 8 seconds to complete. Several factors can contribute to this situation.

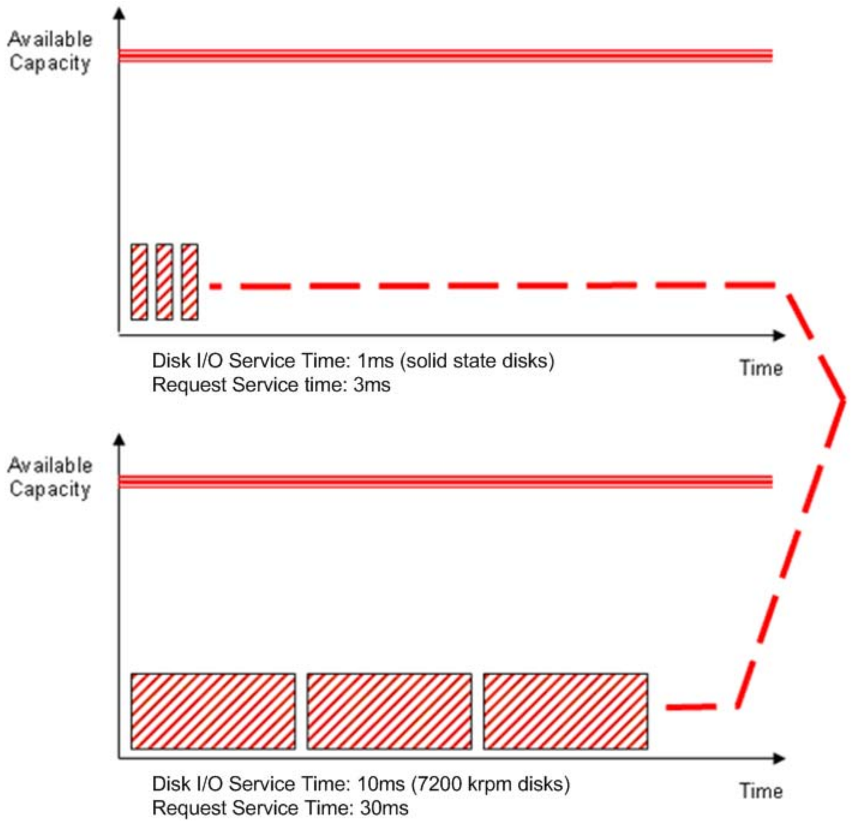
- Many results per page amplifies the amount of processed data. The random nature of results matching the search keyword indicates a low probability that Oracle or the storage array cashes the results, especially if amount of data exceeds the available cache memory at either tier. In the case being considered, the storage array is sized for 50,000 disk I/O per second, but is only servicing about 800 physical disk I/Os in 8 seconds, so the storage array is not the problem.
- DQL select statements against custom object types generally execute as NESTED LOOPS with TABLE ACCESS, the queries are not issued against an index.
- The Oracle execution plan is issuing 800 disk I/Os serially. Each I/O request runs against a 7200 RPM drive with 0.01-seconds service time per request. As a result, 800 physical disk I/Os issued serially takes 800 * 0.01 = 8 seconds.

To improve the performance, do any or all of the following:

- Do less. Make the application retrieve less than 100 rows per page. Retrieving 50 rows per page instead of 100 reduces the amount of data processed by 50% and reduces the service time by the corresponding amount.
- Do something different. Create "covering indexes" and influence the execution plan to replace TABLE ACCESS with INDEX ACCESS. Indexes are much smaller, packed tighter, more likely to be cached, and can fit more easily into memory than a full table row. Getting data from an Oracle buffer cache reduces service times by several orders of magnitude.
- Do it faster. Request more storage array cache and/or lower latency storage. If the Oracle data storage used solid-state drives (SSD) with a latency of 0.001 seconds, the 8-second request takes 0.8 seconds.

Figure 27, page 71 illustrates the affect of latency on throughput.

Figure 27. Single request executing three serial disk I/Os


Available Capacity
Disk I/O Service Time: 1ms (solid state disks)
Request Service time: 3ms
Time
Available Capacity
Disk I/O Service Time: 10ms (7200 krpm disks)
Request Service Time: 30ms
Time

Do not confuse latency for throughput. A query that runs in less than 1 second on a laptop can generate 10,000 logical I/Os per execution on a production system if 1,000 concurrent users run it every few seconds. A quick calculation shows that 1,000 users running the same query that processes 10,000 logical I/Os ends up processing 76 GB of data (1,000 * 10,000 blocks * 8 KB/block). Even if all data blocks are in memory (doubtful) and no CPU is used during the transactions (even more doubtful), since backplanes transfer about 10-20 GB-second between memory and CPU, the transactions take at least 4-8 seconds. In this case, single user latency numbers can be misleading because the capacity is not available to support the transaction under a user load.

Measuring single user performance

To assess single user performance, collect DMCL/DFC traces on unit tests. The traces indicate how much time is spent on Content Server executing the RPC versus the client, and how often a command is called. The performance analyst can determine how much response time is spent on the Content Server in comparison to the application server. Excess time spent issuing RPC commands indicates a Content Server problem. Otherwise, a problem in the client side is indicated.

Single user testing captures DFC traces and helps determine if the problem is in the front of the application server (such as the network, browser, and client PC) or behind the application server (the back-end infrastructure from the application server down to disk drives). Single user testing involves the following steps:

1. Determine the transactions to measure.
2. Prepare a quiet application server.
3. Warm up the system.
4. Turn on DFC tracing.
5. Execute the business transaction.
6. Turn off DFC tracing.
7. Process and analyze your trace files.

Determine transactions to measure — Determining the business transaction and breaking it into distinct and separate steps is critical. There is no rule here other than your choice of what you wish to measure and potentially improve. An example of a business transaction is "Hockey Team Canada just won the gold medal, I must find a picture of the winning goal". This business transaction can be broken into the following steps:

1. Search for keywords "hockey" and "canada" and display list of results.
2. Click the "get me next page" icon.
3. Click image to view properties.
4. Export the image to local drive for publication composition tools.

Prepare a quiet application server — Use a quiet application server to ensure your trace files do not become contaminated by other users of the system. This requirement is especially important when tracing is done on a production system or you are unable to declare the application server as off-limits.

Warm up the system — Warming up the system is like warming up the Oracle library cache. Warming up the system makes sure one-time activities (caching of the BOF jar files, for example) do not affect actual measurements. It is also critical that you use the correct data to avoid (or not) caching at either the database layer (the Oracle buffer cache, for example) or the storage array layer (the vmax cache, for example). For example, you can search for one keyword during warm-up and another when collecting traces. This way, your results are unlikely to be in the database or storage layer cache, unless you want to test response times when data is cached at these layers.

Turn on DFC trace — Turn on the DFC trace file collection by setting the following in `dfc.properties` file:

```
# tracing
dfc.tracing.enable=true
dfc.tracing.file_prefix=dam_prefix
dfc.tracing.include_rpcs=true
dfc.tracing.include_session_id=true
dfc.tracing.max_stack_depth=0
dfc.tracing.mode=compact
dfc.tracing.timing_style=seconds
dfc.tracing.verbose=true
dfc.tracing.verbosity=verbose
```

Execute the business transaction — After the log file (specified as dam_prefix in the preceding `dfc.properties` file) shows up in the logs directory, run your transaction. After completion of the transaction, turn off DFC tracing and make a copy of the trace file.

Turn off DFC tracing — Turn off DFC tracing by disabling tracing, as shown in the following in `dfc.properties` file:

```
# tracing
dfc.tracing.enable=false
dfc.tracing.file_prefix=dam_prefix
dfc.tracing.include_rpcs=true
dfc.tracing.include_session_id=true
dfc.tracing.max_stack_depth=0
dfc.tracing.mode=compact
dfc.tracing.timing_style=seconds
dfc.tracing.verbose=true
dfc.tracing.verbosity=verbose
```

Determining RPC call source

To determine the location of RPC calls, configure your <*TaskSpace home*>`\WEB-INF\classes\log4j.properties` and `dfc.properties` files to have the stack trace of the call sent to the *$DOCUMENTUM*`\log4j.log` file as follows:

1. Turn on debugging in your `log4.properties` file as follows:

   Note: You can use the following listing for all your log4j settings.

   ```
   log4j.rootCategory=DEBUG, Console, File
   log4j.category.MUTE=OFF
   log4j.additivity.tracing=false

   log4j.appender.Console=org.apache.log4j.ConsoleAppender
   log4j.appender.Console.threshold=WARN
   log4j.appender.Console.layout=org.apache.log4j.PatternLayout
   log4j.appender.Console.layout.ConversionPattern=%d{ABSOLUTE} %5p [%t] %m%n

   log4j.appender.File=org.apache.log4j.RollingFileAppender
   log4j.appender.File.File=C:/documentum/logs/log4j.log
   log4j.appender.File.MaxFileSize=100MB
   log4j.appender.File.layout=org.apache.log4j.PatternLayout
   log4j.appender.File.layout.ConversionPattern=%d{ABSOLUTE} %5p [%t] %m%n
   ```

2. Modify your `dfc.properties` file as follows:

   Note: A large `max_stack_depth` is not necessary.

   a. Update the application server `dfc.properties` file to enable tracing.

   b. Set the `method_name_filter` to match the method calls to be captured.

   c. Disable DEBUG messaging for the filter of interest by setting the log category to a string not configured for debugging in the `log4j.properties` file.

   For example:

   ```
   dfc.tracing.enable=true
   dfc.tracing.verbose=true
   ```

```
dfc.tracing.include_rpcs=true
dfc.tracing.max_stack_depth=1
dfc.tracing.method_name_filter=*.applyForObject()
dfc.tracing.print_stack_on_method_match=true
dfc.tracing.log.category[0]=garbage
```

3. View the `log4j.log` file. Stack traces are generated for matching filters.

```
20:40:27,214 DEBUG [main] com.documentum.fc.client.impl.connection.
docbase.netwise.NetwiseDocbaseRpcClient.applyForObject() Entered.
The current call stack is:
java.lang.Exception: EXCEPTION TO GET CALL STACK
 at com.documentum.fc.client.impl.connection.docbase.netwise.
NetwiseDocbaseRpcClient.applyForObject(NetwiseDocbaseRpcClient.java)
 at com.documentum.fc.client.impl.connection.docbase.
DocbaseConnection$8.evaluate(DocbaseConnection.java:1246)
 at com.documentum.fc.client.impl.connection.docbase.DocbaseConnection.
evaluateRpc(DocbaseConnection.java:1014)
 at com.documentum.fc.client.impl.connection.docbase.DocbaseConnection.
applyForObject(DocbaseConnection.java:1238)
 at com.documentum.fc.client.impl.docbase.DocbaseApi.
parameterizedFetch(DocbaseApi.java:106)
 at com.documentum.fc.client.impl.objectmanager.PersistentDataManager.
fetchFromServer(PersistentDataManager.java:194)
 at com.documentum.fc.client.impl.objectmanager.PersistentDataManager.
getData(PersistentDataManager.java:92)
 at com.documentum.fc.client.impl.objectmanager.PersistentObjectManager.
getObjectFromServer(PersistentObjectManager.java:355)
 at com.documentum.fc.client.impl.objectmanager.PersistentObjectManager.
getObject(PersistentObjectManager.java:311)
 at com.documentum.fc.client.impl.session.Session.getObjectWithCaching(Session.java:859)
 at com.documentum.fc.client.impl.TypeManager.getType(TypeManager.java:58)
 at com.documentum.fc.client.impl.session.Session.getType(Session.java:1215)
 at com.documentum.fc.client.impl.session.SessionHandle.getType(SessionHandle.java:906)
 at com.documentum.fc.client.DfPersistentObject.getType(DfPersistentObject.java:1297)
 at com.documentum.fc.client.content.impl.storePicker.StoragePolicyHelper.
getStoragePolicy(StoragePolicyHelper.java:145)
 at com.documentum.fc.client.content.impl.storePicker.StoragePolicyHelper.
determineStoreFromStoragePolicy(StoragePolicyHelper.java:30)
 at com.documentum.fc.client.content.impl.storePicker.CheckinStorePicker.
determineStore(CheckinStorePicker.java:35)
 at com.documentum.fc.client.content.impl.ContentManager.
prepareAContentForCheckin(ContentManager.java:1138)
 at com.documentum.fc.client.content.impl.ContentManager.prepareForCheckin
(ContentManager.java:1112)
 at com.documentum.fc.client.DfSysObject.doCheckinImpl(DfSysObject.java:878)
 at com.documentum.fc.client.DfSysObject.doCheckin(DfSysObject.java:834)
 at com.documentum.fc.client.DfSysObject.checkinEx(DfSysObject.java:800)
 at com.documentum.fc.client.DfSysObject.checkin(DfSysObject.java:793)
 at com.documentum.fc.client.DfDocument___PROXY.checkin(DfDocument___PROXY.java)
 at com.documentum.dmcl.impl.CheckinHandler.get(CheckinHandler.java:33)
 at com.documentum.dmcl.impl.DmclApi.get(DmclApi.java:41)
 at com.documentum.dmcl.impl.DmclApiNativeAdapter.get(DmclApiNativeAdapter.java:136)
 at com.documentum.dmcl.impl.DmclApiNativeAdapter.get(DmclApiNativeAdapter.java:122)
```

Analyzing large queries

Collect the trace data and convert it to a Microsoft Excel friendly format using the `traceD6.awk` script. The `traceD6.awk` script is available on the EMC Community Network (https://community.emc.com/docs/DOC-6355).

Analyze the histogram report (Figure 28, page 75). Large queries are easy to identify.

Figure 28. Large query trace

```
****** TRACE_RPC_HIST (D6 VERSION) ****

DURATION (secs):             15.781
TIME SPENT EXECUTING RPCs (secs):  13.620 (which is 86.31 percent of total time)
Threads :                           2
Connections :                           1

****** PROFILE OF rpc CALLS *****



CALL YIELD  % OF TOTAL  AVERAGE TIME  # OF
TIME (secs)  RPC TIME  PER RPC (secs) CALLS   RPC NAME
     0.006        0.0      0.006          1 closeCollection
     0.883        6.5      0.009        100 ObjGetXPermit
     7.981       58.6      0.319         25 EXEC_QUERY
     0.247        1.8      0.002        100 ObjGetPermit
     1.868       13.7      0.081         23 multiNext
     2.635       19.3      0.026        101 SysObjFullFetch

**** QUERY RESPONSE SORTED IN DESCENDING ORDER ****


qry rsp query
3.511 select  parent_id, count(parent_id) as rendition_count from dmr_content  where
  ( ANY parent_id IN ('0902218e808a0731',...,'0902218e8010df71') and rendition != 0)
  group by parent_id
3.237 select  parent_id, count(parent_id) as rendition_count from dmr_content  where
  ( ANY parent_id IN ('0902218e8010db86',...,'0902218e800df391') and rendition != 0)
  group by parent_id
0.261 select headline,caption,source,r_object_id from cch_photo (all)  where
  r_object_id IN ('0902218e808a0731',...,'0902218e800df391')
...
```

From the trace, you can see that it takes 6.748 seconds to execute the one query twice with different IDs. These two query executions, with runtimes of 3.511 seconds and 3.237 seconds, account for 85% of the 7.981 seconds spent on all queries, 50% of the 13.620 seconds spent executing RPCs, and 42% of the entire transaction time.

Analyzing process variable usage (measuring fetches)

Assess the number of object fetches in your application by running a DFC trace.
Use the `object_breakdown.awk` script, available on EMC Community Network (https://community.emc.com/docs/DOC-6355), to produce a report that breaks down the types of object fetches. The following provides a sample output of the `object_breakdown.awk` script:

```
****** Total number of object fetches per type *****
```

```
9 Unique Object Types Fetched
58 Total Objects Fetched
Fetches  Type
    1 rim7_record
    1 dm_sysobject
    1 dmc_wfsd_rim_sdt
   23 dmc_wfsd_element_string
    3 dmc_wfsd_type_info
    1 dm_process
   25 dmc_wfsd_element_parent
    1 dmc_wfsd_element_integer
    2 dm_activity

****** Total number of iscurrent calls per type *****
11 Unique Object Types IsCurrent Called
147 Total IsCurrent Calls
Fetches  Type
   23 dmc_wfsd_type_info
    1 dmc_wfsd_rim_sdt
   22 dm_activity
    3 rim7_record
   21 dm_process
   27 dm_workflow
    1 dm_relation
    3 dmi_package
   15 dmi_queue_item
    1 dm_user
   30 dmi_workitem
```

The report shows 25 parent objects being fetched (`dmc_wfsd_element_string`) along with 23 string process variables (`dmc_wfsd_element_string`) and 1 integer process variable (`dmc_wfsd_element_integer`), for a total of 49 fetches. Consolidating the 23 string process variables and 1 integer process variable into one SDT reduces the number of fetches by 46.

Analyzing query results processing

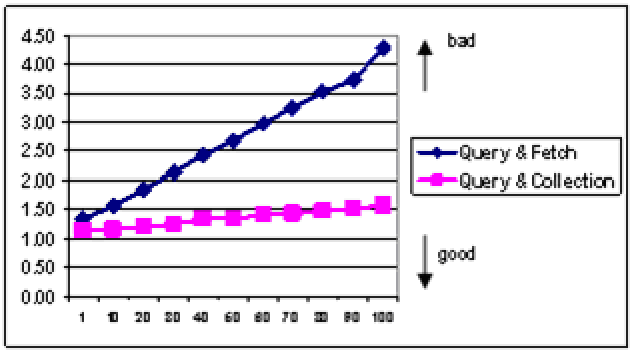
Query results processing adversely impacts performance as the result set size and query usage increase. Analyze this issue by inspecting the raw DFC trace files. Look for the following type of call sequence:

```
...
EXEC_QUERY RPC, e.g. select r_object_id from …
MultiNext RPC (gets up to 50 results)
SysObjectFullFetch (fist object)
SysObjectFullFetch (2nd object)
SysObjectFullFetch (3nd object)
...
```

Figure 29, page 77 shows how service times grow when using query and fetch instead of query and collection.

Note: There can be situations where you cannot query and fetch code pattern.

Figure 29. Result set sizes and service times

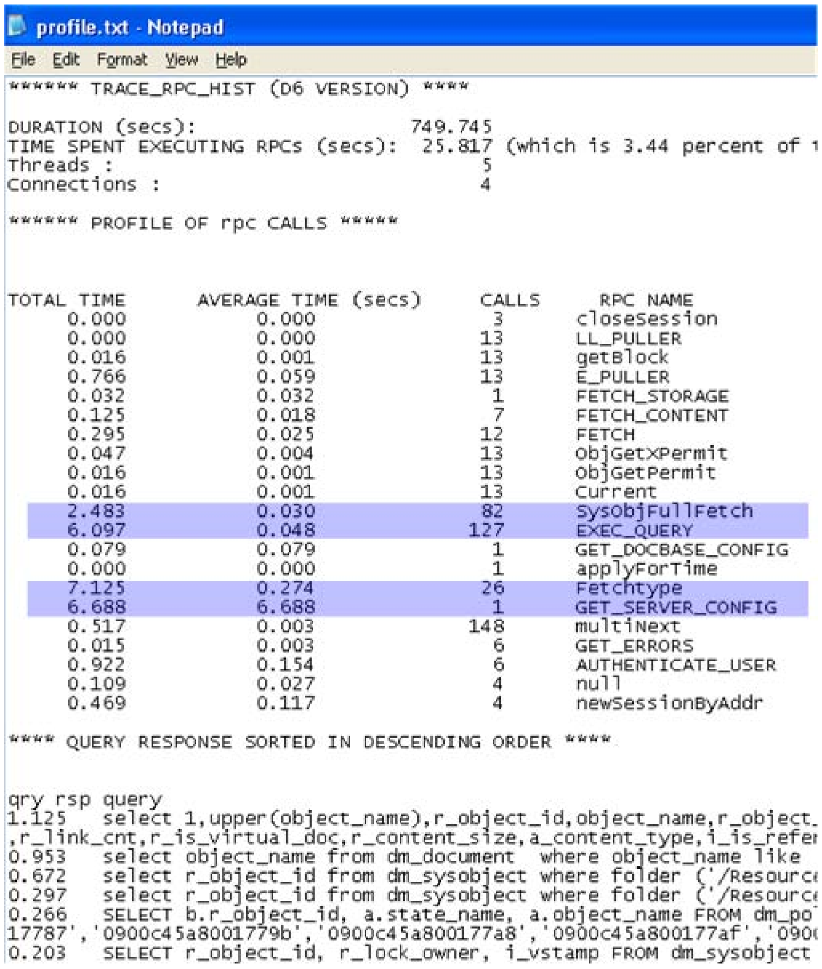


Change the application code to use collections instead of multiple fetches. For example, replace retrievals with `select r_object_id, attr1, attr2, … from …` and replace updates with `update type … set attr1=…, set attr2=…, where ….`

Analyzing many small RPC calls

For many small queries, run a histogram of trace data using the `trace_rpc_histD6.awk` script (Figure 30, page 78). Focus your attention on high yield RPCs (highlighted rows in Figure 30, page 78). Maximizing query yield, page 36 provides information on maximizing your query yield. The `trace_rpc_histD6.awk` script is available on the EMC Community Network (https://community.emc.com/docs/DOC-6355)

Figure 30. Histogram for many small queries



```
****** TRACE_RPC_HIST (D6 VERSION) ****

DURATION (secs):                    749.745
TIME SPENT EXECUTING RPCs (secs):   25.817 (which is 3.44 percent of t
Threads :                                5
Connections :                            4

****** PROFILE OF rpc CALLS *****


TOTAL TIME       AVERAGE TIME (secs)     CALLS     RPC NAME
     0.000            0.000                 3     closeSession
     0.000            0.000                13     LL_PULLER
     0.016            0.001                13     getBlock
     0.766            0.059                13     E_PULLER
     0.032            0.032                 1     FETCH_STORAGE
     0.125            0.018                 7     FETCH_CONTENT
     0.295            0.025                12     FETCH
     0.047            0.004                13     objGetXPermit
     0.016            0.001                13     objGetPermit
     0.016            0.001                13     Current
     2.483            0.030                82     SysObjFullFetch
     6.097            0.048               127     EXEC_QUERY
     0.079            0.079                 1     GET_DOCBASE_CONFIG
     0.000            0.000                 1     applyForTime
     7.125            0.274                26     Fetchtype
     6.688            6.688                 1     GET_SERVER_CONFIG
     0.517            0.003               148     multiNext
     0.015            0.003                 6     GET_ERRORS
     0.922            0.154                 6     AUTHENTICATE_USER
     0.109            0.027                 4     null
     0.469            0.117                 4     newSessionByAddr

**** QUERY RESPONSE SORTED IN DESCENDING ORDER ****


qry rsp query
1.125   select 1,upper(object_name),r_object_id,object_name,r_object_
,r_link_cnt,r_is_virtual_doc,r_content_size,a_content_type,i_is_refe
0.953   select object_name from dm_document  where object_name like
0.672   select r_object_id from dm_sysobject where folder ('/Resourc
0.297   select r_object_id from dm_sysobject where folder ('/Resourc
0.266   SELECT b.r_object_id, a.state_name, a.object_name FROM dm_po
17787','0900c45a8001779b','0900c45a800177a8','0900c45a800177af','090
0.203   SELECT r_object_id, r_lock_owner, i_vstamp FROM dm_sysobject
```

Measuring adaptor performance

There are two ways to understand the performance impact of your adaptors.

- Create a baseline (use DFC tracing, firebug, or a reverse proxy) and take away the adaptors 1 by 1. Measure the improvement on form initialization after taking away all adaptors. This approach provides you with the incremental cost of adding adaptors to the form.
- To isolate adaptor query performance from performance issues related to the containing form, copy the query for each adaptor and run the query in IDQL or DA. Observe the response time. Poor query performance can result from network latency or poorly written queries. The same rules that apply to poorly written SQL or DQL apply to JDBC adaptors that query external sources. Adaptors perform poorly if the adaptor queries contain leading wildcard filters or return large amounts of data.
  — Make the DQL query in your adaptor more selective.

```
[[i.e]&linkCreation=true&fromPageID=21234961"
   class="createlink"linktype="raw"linktext="[[i.e]|[[i.e]">
   [[i.e]&linkCreation=true&fromPageID=21234961"
   class="createlink"linktype="raw" linktext="[i.e]|[i.e]">
   [i.e]|[i.e]] it should return as few rows as possible.
   Current Query select distinct sender from
   [[dbo.vw_sender]&linkCreation=true&fromPageID=21234961"
   class="createlink"linktype="raw" linktext="[[dbo.vw_sender]|[[dbo.vw_sender]">
   [[dbo.vw_sender]&linkCreation=true&fromPageID=21234961"
   class="createlink"linktype="raw" linktext="[dbo.vw_sender]|[dbo.vw_sender]">
   [dbo.vw_sender]|[dbo.vw_sender]] where country like '%$
```

```
Unknown macro: {country}
```

```
' order by 1 Change the query to select distinct sender
   from [[dbo.vw_sender]|[dbo.vw_sender]] where country = '$
```

```
' order by 1 Or the other option is to select a country by default
```

— Design adaptor queries to take advantage of available indexes or create indexes to accommodate the DQL query.

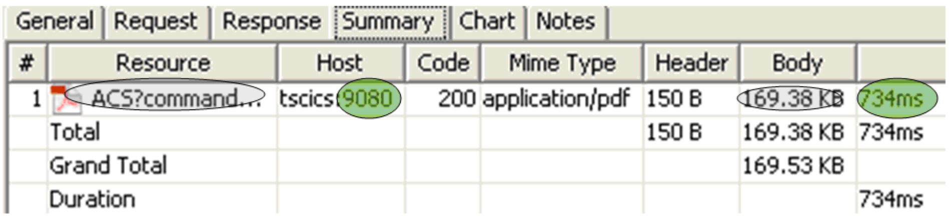
Checking ACS operation

Check ACS operation as follows:

- Verify the ACS URL.
- Verify the ACS port.
- Observe the amount of data sent.
- Observe the time it takes to view a document.

Use a reverse proxy to ensure the ACS URL comes from the Java Method Server and includes ’ACS’ in the URL (Figure 31, page 79).

Figure 31. Verifying the ACS URL and port

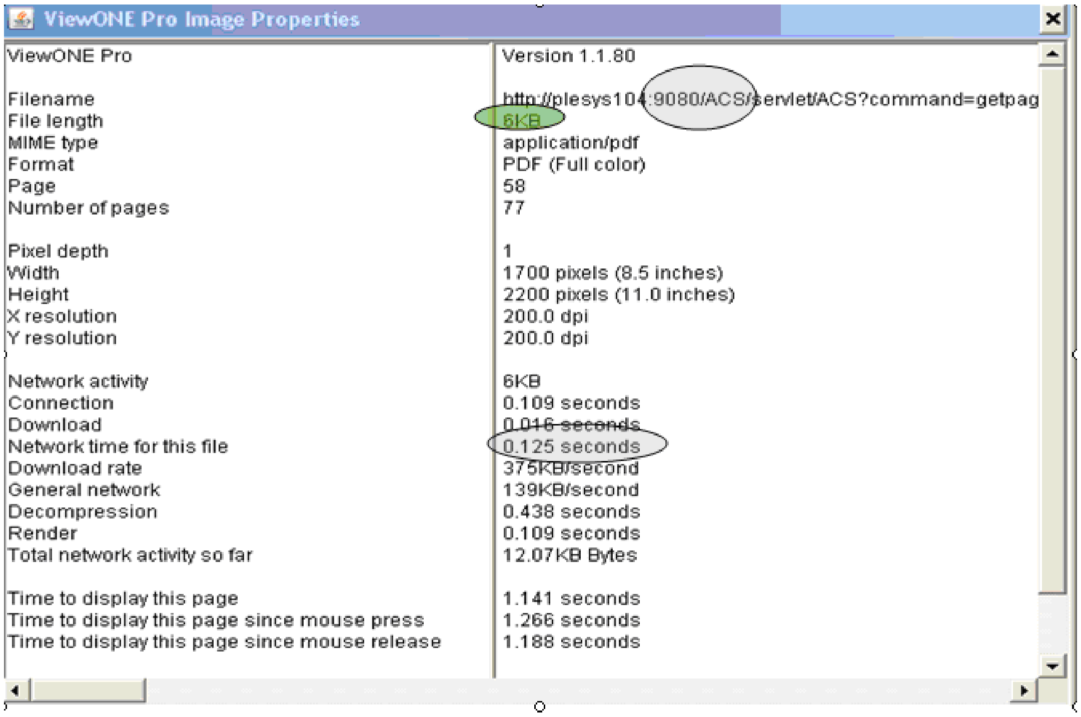


#	Resource	Host	Code	Mime Type	Header	Body	
1	ACS?command...	tscics 9080	200	application/pdf	150 B	169.38 KB	734ms
	Total				150 B	169.38 KB	734ms
	Grand Total					169.53 KB	
	Duration						734ms

View the image properties (Figure 32, page 80) and note the following:

- File URL contains the ACS port and the word ’getpage’.
- Page viewing time is reasonable.
- File length is in kilobytes.
- Network time and download rate rule out network issues.

Figure 32. ACS image properties in Daeja



ViewONE Pro	Version 1.1.80
Filename	http://plesys104:9080/ACS/servlet/ACS?command=getpag
File length	6KB
MIME type	application/pdf
Format	PDF (Full color)
Page	58
Number of pages	77
Pixel depth	1
Width	1700 pixels (8.5 inches)
Height	2200 pixels (11.0 inches)
X resolution	200.0 dpi
Y resolution	200.0 dpi
Network activity	6KB
Connection	0.109 seconds
Download	0.016 seconds
Network time for this file	0.125 seconds
Download rate	375KB/second
General network	139KB/second
Decompression	0.438 seconds
Render	0.109 seconds
Total network activity so far	12.07KB Bytes
Time to display this page	1.141 seconds
Time to display this page since mouse press	1.266 seconds
Time to display this page since mouse release	1.188 seconds

Running multi-user (load) tests

In contrast to single user tests which focus on application design performance regarding transaction latency, multi-user load tests focus on system performance while under the load of a full set of users. Multi-user load testing provides the best way to simulate production environment performance for current and future user volume. Multi-user testing exposes potential problems in system configuration and sizing for the targeted user volume as evidenced by consumption of system resources such as CPU time, memory heap, database connection, and network bandwidth. Ensure the following SLAs are met during your multi-user load tests:

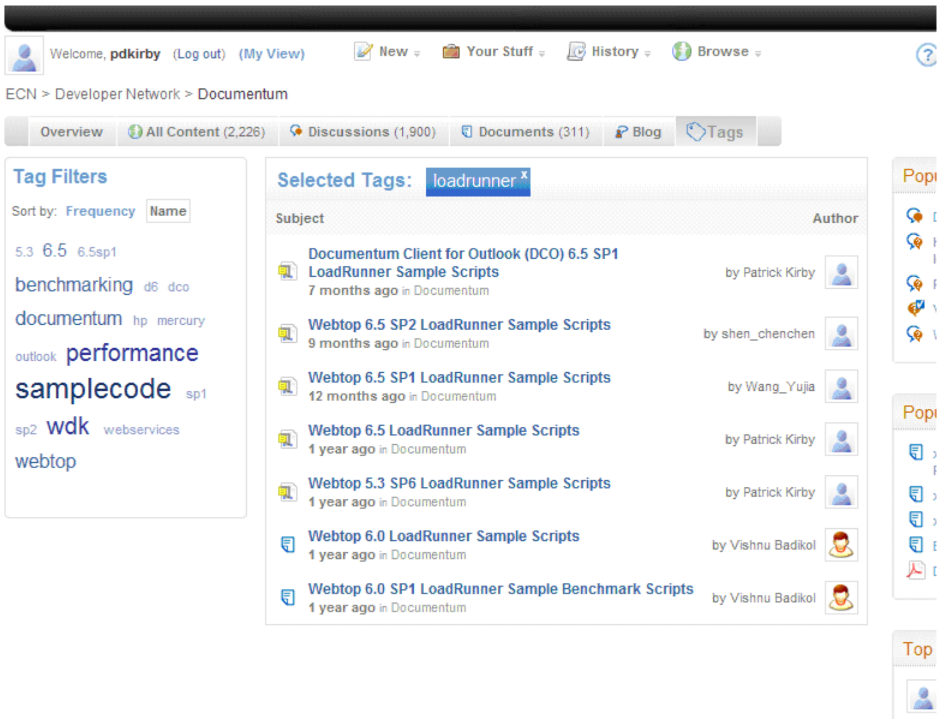
- Operation response times for the end-user community are acceptable.
- Hardware infrastructure provides adequate performance and capacity.
- Wide Area Network (WAN) condition meets expectations.
- Hardware growth expectation matches expected user growth volume.

Note: Do not use multi-user testing for functional testing of individual operations.

Develop an automated test harness — An automated testing harness is essential for efficiently running consistent and repeatable multi-user performance tests, and for collecting performance metrics during the test runs. The EMC Enterprise Engineering Performance Group uses HP Mercury Load Runner and JMeter for all multi-user load testing.

Currently, there are no out-of-the-box xCP Load Runner test scripts available for customer use. The EMC Customer Network (ECN) provides sample Load Runner scripts for the EMC Documentum Webtop product (Figure 33, page 81).

Figure 33. Sample Load Runner testing scripts for Webtop



Subject	Author
Documentum Client for Outlook (DCO) 6.5 SP1 LoadRunner Sample Scripts 7 months ago in Documentum	by Patrick Kirby
Webtop 6.5 SP2 LoadRunner Sample Scripts 9 months ago in Documentum	by shen_chenchen
Webtop 6.5 SP1 LoadRunner Sample Scripts 12 months ago in Documentum	by Wang_Yujia
Webtop 6.5 LoadRunner Sample Scripts 1 year ago in Documentum	by Patrick Kirby
Webtop 5.3 SP6 LoadRunner Sample Scripts 1 year ago in Documentum	by Patrick Kirby
Webtop 6.0 LoadRunner Sample Scripts 1 year ago in Documentum	by Vishnu Badikol
Webtop 6.0 SP1 LoadRunner Sample Benchmark Scripts 1 year ago in Documentum	by Vishnu Badikol

Limit key operations — A few key operations typically characterize system workload. Restrict your scripting to include this small set of key operations, preferably below 10. As user volume rises to 10,000, the number of recorded metrics increases. Recording metrics for many operations and users can result in an uninterpretable volume of metrics.

Pre-load your test environment — Prepare and implement your data loading strategy before running multi-user tests. Populate system data to simulate target user volume and queued tasks during various state of the business process. Populate the repository with documents that are ready for viewing by the application.

Backup and restore your test environment — Prepare and implement a backup and restore strategy before running the multi-user tests. Create a backup of your pre-loaded test environment so that multiple rounds of multi-user testing can be executed from the same starting point by restoring the backup image. Restoring an image saves time because you do not need to repeat your data loading or perform any data cleansing after each multi-user test.

Use an isolated near-production like environment — Conduct load tests on an isolated near-production like environment. An isolated environment makes analysis of performance metrics more valid as there are no confounding variables to deal with. Using a near-production like environment increases the likelihood of finding the right level of configuration parameters for production use.

Slowly ramp-up user volume — Slowly ramp-up user volume during multiple test runs. Data captured during the slow ramp up process provides important information for calculating and projecting capacity utilization. Ramp-up user load in a realistic fashion; such as one user per second.

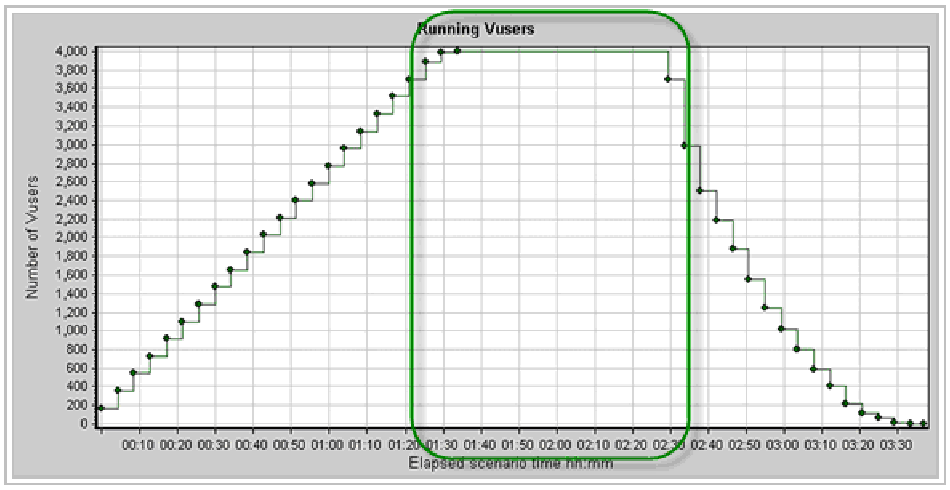
Build in realistic idle time — Provide realistic idle time between user operations. Base your built-in idle time on expectations of real-user behavior. Real-user delays between successive operations result from "think time", coffee breaks, or other forms of distraction. Your testing scenario can bundle many concurrent users to create high aggregate volume of incoming operation requests. However, the better you can model each user session with the appropriate between operation delays, the better your multi-user testing can approximate performance expectations for the production system.

Monitor coarse grain metrics — Setup monitors to record coarse grain performance metrics on all hardware. Bottlenecks can occur in any tier of the hardware infrastructure. Conduct a more detailed analysis of those areas indicated as a problem by the coarse grain performance metrics. The coarse grain performance metrics are:

- CPU consumption of the servers
- memory consumption of the servers
- network bandwidth consumption
- response time of a few key operations
- I/O activities of the servers or storage.

Run peak volume in steady state — Run each multi-user test and record the result for a steady state period of at least 30 minutes. Keep each multi-user test run within several hours, including the ramp-up time for adding users. Figure 34, page 82 illustrates a multi-user test in which it took 1.5 hours to ramp-up 4,000 users, which then ran at the 4,000 user steady state for one hour.

Figure 34. Ramping up number of users



Analyzing multi-user tests

After completing your multi-user tests, analyze the following areas for bottlenecks:

- Determine which server or process consumed the most CPU or memory.
- Check the database server for queries that take a long time, deadlock or core dump alerts, and top wait events. For Oracle user, use an AWR report for your analysis (See Assessing database bottlenecks and dead waits, page 84).
- Check the J2EE layer for excessive garbage collections, which indicates a bottleneck in the application server. If so, add additional JVM instances to improve load balancing.
- Check for bottlenecks indicated in the Content Server repository and BPM logs, and the application server logs.
- Check for network bottlenecks indicated by bandwidth exhaustion or by exceeding the maximum number of packets per second that can be transferred. Even though network bandwidth is not exhausted, a typical network card can only handle definite number of packets a second. In this case adding additional network cards can help.

When addressing a problem area, incrementally make one change at a time so that you can understand the impact of the change, then rerun the test. Depending on the result of the rerun test, you can either apply or reject your change.

Avoiding multi-user test worst practices

The following provides activities to avoid during multi-user testing:

- Do not turn on excessive logging or tracing. The volume of log files generated in a multi-user load test makes analysis of results difficult.
- Do not synchronize users to execute an operation at the same time. Interpretation of these types of results regarding the true capability of your system can be misleading.
- Avoid simulating super-users. Using a few super-users to process requests is not the same as spreading the same number of requests over a larger set of normal-users. For the normal-user scenario, each user creates a unique backend session with caching and history being tracked under that particular session. The purging and reuse of cache data is based on the wall-clock request time of that user session. A super-user session violates most of the built-in design algorithms because request times are unrealistically condensed.
- Do not exceed 80% CPU usage or memory capacity on your servers. During normal usage, operating systems require 20% excess capacity to handle naturally occurring usage spikes. Red-lining servers generates results that are misleading and bring no value to the overall validation of the system. Be prepared to add servers when 80% capacity is reached.
- Avoid running multi-user testing on unstable builds. Make sure your single user testing is acceptable before doing multi-user testing.

Assessing database bottlenecks and dead waits

Bottleneck (Assessing capacity bottlenecks, page 84) and dead waits (Assessing dead waits, page 84) indicate system sizing issues.

Assessing capacity bottlenecks

Capacity bottlenecks involve points in the system waiting for disk I/O to complete. For Oracle databases, the DBWR (Database Writer) is a single point of serialization. Disk I/O directly relates to the number of redo logs generated.

Logical I/O (LIO) can easily become Physical I/O (PIO) in a large volume system. At a certain point, you can no longer cache.

A full index scan of 10 million table rows, with index that has 256 bytes per row, still reads 2.3 GB of index data, which puts demands on resources (CPU, memory, disk I/O).

- Look for heavy LIO/PIO service activity in the database statspack.
- Look at AUTOTRACE execution paths and statement statistics for suspect queries.
- Figure out how often LIO/PIO statistics get run with number of users on the system and extrapolate the I/O load on the system.

Assessing dead waits

Dead waits occur when multiple processes (or users) try to update a locked database row, resulting in a race condition. Asynchronous event handling can also cause race conditions, even if your process design effectively partitions the workload.

Carefully design your process to partition the workload and ensure workload partitioned updates do not conflict with agent-based updates.

Chapter 8

Maintaining the Repository and BAM Databases

This chapter provides guidelines for maintaining the system databases and includes the following topics:

xCP applications frequently query databases for process data. Over time, the database tables can grow significantly, which affects query and application performance. Periodic maintenance of the repository and BAM databases can help keep application performance at the same levels as when the application was new.

Maintaining the repository database

Periodic database maintenance can prevent significant and sudden reductions in system performance (throughput or response time). The following workflow processing and task lookup tables cause roughly 80% of database performance problems.

- `Dm_workflow_s`
- `Dm_sysobject_s`
- `Dm_process_s`
- `Dmi_workitem_s`
- `Dmi_queue_item_s`

These tables usually have many inserts and deletes and can become quickly fragmented. If you notice performance issues, examine the statistics for the table indexes. Pay close attention to the `dmi_queue_item_s`, which can quickly grow into millions of rows. Existence of any entries that have the user name `dm_fulltext_index_user` indicates a problem.

Note: If your system does not use the embedded xCP full-text search server, disable the system from writing entries to the `dmi_queue_item_s` table.

If the repository does not use an index agent, prevent Content-Server from generating queue items for the user name `dm_fulltext_index_user` as follows:

1. Go to `$DM_HOME/install/tools` folder on the Content Server host.
2. Run the `unresigertype.ebs` script to unregister the events for `dm_sysobject`:

   ```
   cmd>dmbasic -funregistertype.ebs -emain1 - <docbase> <username>
   <password> dm_sysobject.
   ```

To delete existing queue items created for full-text user, run the following:

```
API>query,c,delete dmi_queue_item objects where name = 'dm_fulltext_index_user'
```

If creating many groups and assigning users to those groups, pay close attention to the following tables:

- `Dm_group_r`
- `Dm_group_s`

Note: Keep user group membership under 1,000 (Managing group memberships, page 53).

Pay close attention to the following workflow-related tables:

- `Dmc_wfsd_type_info_s`
- `Dmc_wfsd_type_info_s`
- `Dmc_wfsd_element_parent_s`
- `Dmc_wfsd_element_s`
- `Dmc_wfsd_`<the SDT name in your workflow>
- `Dmc_wfsd_element_string_s`

Pay close attention to the `dm_audittrail` table. Make sure that there is some archiving strategy for the audit trail table.

See the *Optimizing Oracle for EMC Documentum Best Practices Planning Guide* (http://powerlink.emc.com/km/live1//en_US/Offering_Technical/White_Paper/h5962-optimizing-oracle-for-documentum-wp.pdf) for tuning Oracle databases.

Maintaining the BAM database

In production environments, database administrators monitor and periodically adjust the BAM database table space using the tools provided by the database vendor. If necessary, BAM database tables can be purged.

Indexing the database

Create indexes for SDT and custom entities. Copy the PRS report queries, for the SDTs and custom entities requiring indexing, and send these report queries to the DBA for optimization. Index mandatory filters and date fields. Limit the number of indexes to 6-7 per database table.

Note: BAM indexes out of the box entities such as 'Process Execution'.

Purging and archiving the database

Performance problems can result from BAM database tables that grow quickly in high volume applications. The rate data accumulate depends on the number of active processes, the number of auditable events, and the number of data objects monitored by BAM, written to the BAM database and extracted for the BAM dashboard.

Develop a strategy for purging the BAM database. What to purge and how often to purge depends on how quickly the BAM database fills up and how long data must be retained for reporting purposes. (See the *BAM Implementation Guide* for details.)

Note: Properly indexed tables can retain more data before affecting performance and requiring maintenance.

Applying retention policies

Define a retention policy to archive or purge historical data from the database. The DBA manually executes a retention policy or sets the retention policy to execute automatically by scheduling jobs. The retention policy for aggregation tables can be less aggressive than for execution tables as aggregation tables grow more slowly.

Purging the entire database

Periodically generate a dump file of the BAM database, then purge the entire BAM database. If necessary, historic data can be restored. Depending on business need and the volume of process data, create a snapshot (a dump file) of the BAM database every three or six months.

Purging selective portions of the database

Selectively purge large execution and aggregation tables of data that operational reports no longer need. The BAM database provides instance level tables and nine types of aggregation tables. Over the course of a single day, for example, the 5-minute aggregation table can hold 288 rows, 2016 rows over a week, and 105,210 rows over a year. After a few months, the 5-minute aggregation data becomes irrelevant, so purging this table every six months make sense. The same can be said of the 15-minute aggregation table, although this table holds less data than the 5-minute aggregation table, with 35,040 rows per year compared to 105,210 rows for the 5-minute aggregation table.

Chapter 9

Troubleshooting

This chapter provides a listing of the most common performance-related problems known to occur and links to the topics in the main body of the document that address the issue.

Symptom	Cause (Action)
Too long to open form	Problem with form adaptor(s) (Using adaptors, page 52).
	Form uses process variables instead of SDTs (Designing the process object model (using structured datatypes), page 37).
	Form is too large (Minimizing form size, page 40).
	Problem with document viewing (Embedding documents (document viewing), page 51).
	Form embeds audit trail information (Using task history (audit trail), page 51).
Too long to view document	Problem with document viewing (Embedding documents (document viewing), page 51).
Too long to execute get next task	Work queues, skill set matching, and priorities not designed properly (Designing task lists (work queues), page 47).
	Problem with get next task procedure (Troubleshooting the get next task function, page 49).
Too long to render task list	Too many results returned from task list query (Using search restrictions, page 45).
	Search criteria not specific enough (using case insensitive partial keyword search) (Using search criteria, page 43).
	Too much processing on each row in task list (Rendering task lists, page 49).
	Task list query filters and sorts across multiple large database tables (Changing the default date attribute for sorting a task list, page 51).

Symptom	Cause (Action)
	Drag and drop is enabled (Disabling drag and drop, page 63).
	Custom preconditions are set on each row in task list (Avoiding unnecessary preconditions, page 50).
Too long to see workflow task after completion of activity	In-between (automated) activities take too long (Configuring the workflow agent (polling), page 29).
	Too many activities ahead of you (Minimizing and consolidating activities, page 27).
	Polling interval too long or too short (Configuring the polling interval, page 31).
	Wrong number of workflow threads or workflow agents (Increasing workflow threads and adding Content Servers, page 28).
Too long to complete many end-to-end workflows (not enough throughput)	Single workflow instance takes too long to complete (Maximizing throughput across all workflow instances, page 28).
	Manual activity workload not partitioned properly (Avoiding manual bottlenecks, page 32).
	Wrong number of workflow threads or workflow agents (Increasing workflow threads and adding Content Servers, page 28).
	System not sized properly (Increasing workflow threads and adding Content Servers, page 28).
Document search takes too long	Too many results returned from query (Restricting advanced search results, page 46).
	Search criteria not specific enough (using case insensitive partial keyword search) (Using search criteria, page 43).
	Querying across multiple database tables (Designing the process object model (using structured datatypes), page 37.
	Drag and drop is enabled (Disabling drag and drop, page 63).
	Custom preconditions are applied to query results (Avoiding unnecessary preconditions, page 50).

Symptom	Cause (Action)
BAM dashboard takes too long to load	Publishing too much metadata to audit trail and BAM database (Planning and testing, page 57).
	Dashboards and reports not designed for performance (Designing high performance reports, page 60).
	BAM database not configured (optimized) for metrics reported in BAM dashboard (Maintaining the BAM database, page 86).
BAM dashboard not updated fast enough	Business data reported in BAM dashboard is only updated when an activity completes (Reporting on intra-activity events, page 57).
	BAM database and audit trail synchronization issue (Synchronizing the BAM database, page 57).
	Problems with BAM database capacity (Maintaining the BAM database, page 86).
Performance degrades	Database not maintained (Chapter 8, Maintaining the Repository and BAM Databases).
	System not sized to meet growing demands (Sizing the system, page 22).
Inconsistent performance over time	System sized for average not peak loads (Planning for peak or average loads, page 15).
	System workload not balanced (jobs, human work) (Balancing system load, page 13).
	Individual user activity puts excessive demands on system (Preventing high load user actions, page 36).
	System sharing resources with other applications (Avoiding resource sharing, page 14).
Inconsistent performance for different users	Some users belong to too many groups (Managing group memberships, page 53).
	Client is sharing system resources with other applications (Avoiding resource sharing, page 14).

Symptom	Cause (Action)
	Skill set matching, priorities, and queues not configured properly (Designing task lists (work queues), page 47).

Index