Retail passion. Innovative solutions.

Retek® Extract Transform and Load™ 11.2.2

Performance Tuning Guide

Corporate Headquarters:

Retek Inc.
Retek on the Mall
950 Nicollet Mall
Minneapolis, MN 55403
USA

888.61.RETEK (toll free US)
Switchboard:
+1 612 587 5000

Fax:
+1 612 587 5100

European Headquarters:

Retek
110 Wigmore Street
London
W1U 3RW
United Kingdom

Switchboard:
+44 (0)20 7563 4600

Sales Enquiries:
+44 (0)20 7563 46 46

Fax:
+44 (0)20 7563 46 10

Customer Support

Customer Support hours

Customer Support is available 7x24x365 via email, phone, and Web access.

Depending on the Support option chosen by a particular client (Standard, Plus, or Premium), the times that certain services are delivered may be restricted. Severity 1 (Critical) issues are addressed on a 7x24 basis and receive continuous attention until resolved, for all clients on active maintenance. Retek customers on active maintenance agreements may contact a global Customer Support representative in accordance with contract terms in one of the following ways.

Contact Method	Contact Information
E-mail	support@retek.com
Internet (ROCS)	rocs.retek.com Retek's secure client Web site to update and view issues
Phone	+1 612 587 5800

Toll free alternatives are also available in various regions of the world:

Australia	+1 800 555 923 (AU-Telstra) or +1 800 000 562 (AU-Optus)
France	0800 90 91 66
Hong Kong	800 96 4262
Korea	00 308 13 1342
United Kingdom	0800 917 2863
United States	+1 800 61 RETEK or 800 617 3835

Mail

Retek Customer Support
Retek on the Mall
950 Nicollet Mall
Minneapolis, MN 55403

When contacting Customer Support, please provide:

- Product version and program/module name.
- Functional and technical description of the problem (include business impact).
- Detailed step-by-step instructions to recreate.
- Exact error message received.
- Screen shots of each step you take.

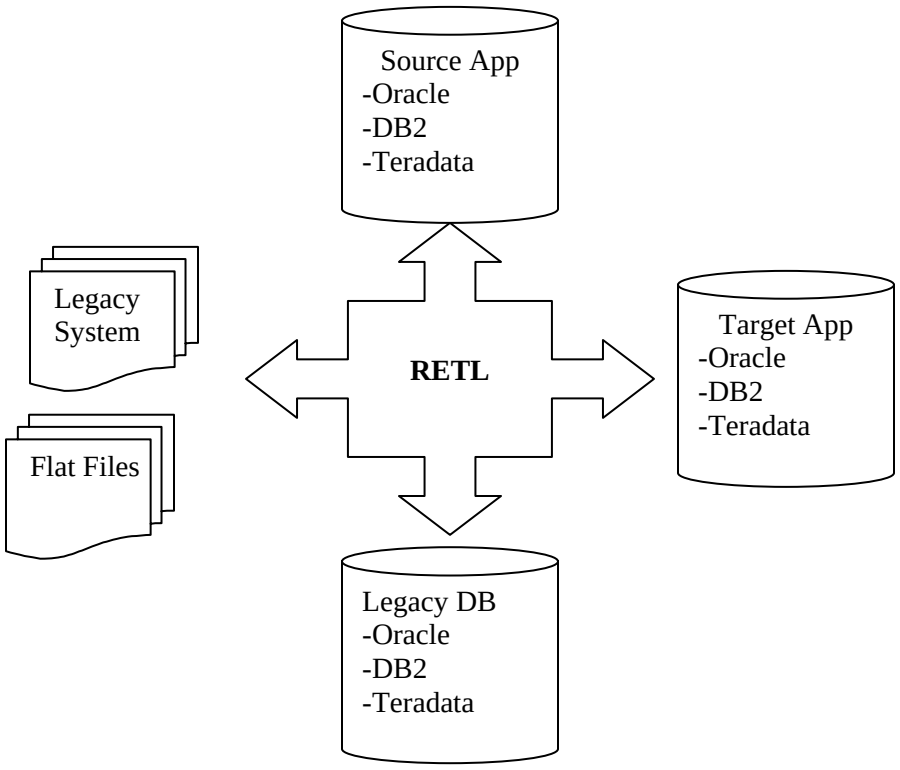
Contents

Chapter 1 – Introduction

Overview

This document came about as a result of the many questions that come from the field about how best to tune and get the most out of the hardware they have, or how to get batch jobs within a certain aggressive time window.

RETL is a simple tool doing a very simple job – moving data. However, tuning and optimizing this data movement process can be very complex and even frustrating because RETL acts as the transport mechanism between different storage mediums and can therefore *seem* to be the bottleneck for certain performance issues. The diagram below shows some of the mediums that RETL can move data between.



In moving data between these locations there are many technologies that are called upon to interact together and can therefore affect the baseline performance of RETL.

A partial list of these is:

1. Hardware
    a. Physical Memory (RAM, cache sizes)
    b. Access Speed of Disks
    c. Throughput of Disks/Controllers
    d. Number of Independent Disks
    e. Bus Speed
    f. Architecture (e.g. 64bits vs 32bits)
    g. Processor Speed
    h. Number of Processors
2. Operating System Software
    a. Version
    b. Architecture
    c. Scheduler
    d. Memory Management
    e. File System
3. Network
    a. Type (Ethernet, Token Ring).
    b. Topology (direct, switched, number of hops)
    c. Speed (10Mps, 100Mbps, Gigabit)
4. Database
    a. Vendor (Oracle, DB2, Teradata)
    b. Version and Patch level of Database
    c. Indexes
    d. Configuration
5. Data
    a. How it is partitioned
    b. Where it is located
    c. How it is related
    d. Number of fields per table
    e. Number of records
    f. Number of tables

6 RETL Based Factors
   a Flow design
   b Buffer size
   c Number of partitions
   d Number of operators
   e Types of operators
   f Version of RETL
7 Java Virtual Machine
   a RETL_VM_MODE
   b JVM Heap Sizing
   c JRE Version
   d Garbage Collection Algorithm

Given that RETL interacts with all of these different elements – a problem in area or subsystem can manifest itself by RETL running slowly – therefore making it look like RETL is slow. This guide is intended to help implementers understand what the complexities are and to give suggestions on how to go about diagnosing problems and issues that might arise in the field.

Note that this document does not attempt to assign levels of importance to each dependency.

This is a living document that will grow over time as we learn more about our customers needs and can fill it with tips from experts from the different domains that RETL touches upon.

Note: The RETL performance tuning suggestions in this document apply to all RETL 11.x releases, unless stated otherwise.

Prerequisites

This document is written toward either an experienced flow developer looking to get the most out of RETL, a core RETL developer or an experienced system administrator. Therefore there is a base level of knowledge that the reader should be equipped with in order to make sense of this guide.

Here are a few of the things you should know (or be prepared to learn) and have if you dive into this document:

- Unix experience – the assumption is that this tuning is taking place on a Unix system.
- Root Access – (or access to someone who does have it) in some cases we will need to make system level changes that may need to be done by/as root.
- Database Admin Access – you will need administrative privileges and/or access to someone who does have administrative access to your database. Parts of this document deals with tuning the database.
- RETL Experience – you've written RETL flows and are very familiar with the RETL Programmer's Guide.

Chapter 2 – General Performance/Tuning Overview

Diagnosing the problem

Understand the Business Requirements

The single most important thing to understand before embarking on Tuning and Optimizing a RETL application is to understand the business requirements. In short, everyone including implementers, project leadership, and end customer should agree on the core requirements.

Generally when we are talking about RETL and Business Requirements we are talking about running a certain key flow (or set of key flows) within a very specific batch window with a worst case set of data. The RETL process must complete within this time window to meet the requirement.

Your aim should be to tune the product until you are able to reach the business requirement – no more, no less. If all involved parties are not comfortable with this – chances are that you have not correctly identified the business requirement. Perhaps the customer would like some breathing room because they anticipate future growth. Whatever the case – make sure everyone is comfortable with when this exercise should stop.

When you achieve consensus, write it down, get buy in on it, get everyone to agree and sign on it. This is the most important element in order to ensure that everyone can walk away with a good feeling after the tuning exercise is complete.

Time is of the Essence

Optimizing and Tuning is a delicate and sometimes time consuming process. It involves much investigative work and can consume weeks, months or years. This is why it is so important to Understand the Business Requirements.

Hopefully, you are reading this document as part of the planning phase. You should talk to your product group to determine how much time and effort will be needed to tune your hardware for their flows. This process goes much faster with a professional onsite. If possible arrange to have a Tuning and Optimization specialist onsite to assist with this aspect of product rollout.

Whether a tuning specialist is going to be onsite or not you should communicate with the product group to understand how long this process generally takes. Make sure that they understand whether you will be working with a specialist or not when they are helping you plan. Generally, you are going to want to take twice as much time without a specialist onsite. Make sure you don't skimp on this test. This is the last line of defense – if you don't find system problems here you are going to find these problems in production.

Narrow the Scope of the Problem

As with diagnosing any problem the key here is to be able to narrow the focus of the problem. We've already discussed all of the potential areas that impact RETL performance. This is the list of problem areas that you will start with. In order to solve the problem then the key is to be able to systematically eliminate possibilities until that list contains a manageable number of elements (maybe 2 or 3).

When you find a performance problem with a RETL flow – attempt to find the simplest flow where the problem exists. If it is in only some flows, but not all – what are the differences in terms of resources (database and flat files) that might point to the cause of the problem? Can you make the flow smaller and simpler to help in diagnosis? Create a list of the operators used in the flow and use the graphviz/print-graph (-g) option to create a visual representation of the flow.

Can you use any tools (see Common Unix Tools below) to help narrow the problem further in terms of system resources being consumed? Is the process I/O bound, CPU bound, simply sleeping? There is always something preventing the flow from running faster.

Tools to diagnose the problem

Web Resources

http://www.google.com/

One of many available online search engines. This one is a fast bare bones, no-nonsense search engine. If you aren't comfortable with search engines you are at a grave disadvantage – learn to use it. It is how much of the research for this document was conducted and will help you in real world situations. Before you call support or anyone for that matter – see if you can find the answers on the web, or at least get some additional ideas to try.

Common Unix Tools

man

In Unix the way to get help on a particular command. Note that you can find most man pages online by typing man <commandname> on almost any search engine.

ps

ps gives a snapshot of the current processes CPU utilization, memory consumption, process ID, parent process ID and some other statistics.

top/prstat/nmon

Tools that show memory/disk/cpu utilization. May not exist on all platforms

vmstat

vmstat, provides a wealth of performance information about virtual memory, process, disk, trap, and CPU. Using this tool, you can locate the source of some common bottlenecks. Remember that when run with no arguments or with interval arguments, the initial output is a summary of all the data since the system was booted.

iostat

The iostat command is used for monitoring system input/output device loading by observing the time the physical disks are active in relation to their average transfer rates. The iostat command generates reports that can be used to change system configuration to better balance the input/output load between physical disks. The iostat command can be run via the following:

iostat –xn 8

This will poll the kernel every 8 seconds for file activity and will present the readings in columnar format by device id. Disk activity should be closely monitored and if any disks are utilized too much, it is recommended to spread activity across other disks. In RETL, this is most easily accomplished by specifying additional TEMPDIRs in rfx.conf and increasing the degrees of parallelism in your RETL flows.

sar

The sar command writes to standard output the contents of selected cumulative activity counters in the operating system. The accounting system, based on the values in the count and interval parameters, writes information the specified number of times spaced at the specified intervals in seconds.

You can select information about specific system activities using flags. The default version of the sar command (CPU utilization report) might be one of the first facilities the user runs to begin system activity investigation, because it monitors major system resources. If CPU utilization is near 100 percent (user + nice + system), the workload sampled is CPU-bound.

mpstat

The mpstat command writes to standard output activities for each available processor, processor 0 being the first one. Global average activities among all processors are also reported. The mpstat command can be used both on SMP and UP machines, but in the latter, only global average activities will be printed.

truss | tusc | strace

Sometimes when you don't know what a process is doing, or perhaps why it failed you can get information on what that process was doing just before it failed. For a given process or command these tools produce a trace of the system calls performs, signals received, and machine faults incurred. Each line of the trace output reports either the fault or signal name or the system call name with its arguments and return value(s). System call arguments are displayed symbolically when possible using defines from relevant system headers; for any pathname pointer argument, the pointed-to string is displayed.

AIX Tools

nmon

nmon is a very handy tool that presents a wide range of performance information such as cpu/memory/disk utilization. AIX systems don't normally have tools such as top/prstat so nmon comes in handy.

uptime

instfix –i

truss

As per the section above "truss | tusc | strace" – truss can usually be found on/for AIX.

HP-UX Tools

prospect

GlancePlus

Hpjmeter

tusc

As per the section above “truss | tusc | strace” – tusc can usually be found on HP-UX. For more information see displaying debugging and performance tuning information with tusc.

Solaris Tools

perfmeter

perfmeter is an OpenWindows XView utility that displays performance values (statistics) for a given hostname. If no host is specified, statistics on the current host are metered. The rstatd(1M) daemon must be running on the machine being metered.

proctool

A GUI-based tool that greatly extends the "top" concept.

truss | strace

These tools as per the section above “truss | tusc | strace” can usually be found on Solaris.

pstack

Pstack gives a dump of the threads currently running

RETL-based diagnostics

Performance profiler and logging

RETL offers performance logging capabilities and a new profiling output (as of RETL 11.2) that greatly aids in diagnosing slow-performing flows. See the RETL Programmer’s Guide about how to use the log file in debugging performance bottlenecks.

Chapter 3 – Tuning the dependencies that affect RETL

In this section, we go into a bit more detail on how each of the dependencies we talked about in the introduction affect RETL and/or general performance, and how to tune these dependencies so that RETL runs optimally.

Hardware

When running a high performance tool you need high performance hardware. There are many subsystems of the hardware that can act as the bottleneck to RETL performance.

Physical Memory (RAM, cache sizes)

As a data movement tool RETL is very dependent upon memory. More RAM means that RETL can hold more data in memory to perform transformations etc before writing to disk and/or the operating system begins swapping to and from disk.

NEVER allow the o/s to page to disk. Paging to disk is extremely detrimental to performance. Starting with the 11.x versions, RETL is written in java, whose memory garbage collector is severely impaired when having to reclaim memory that has been paged to disk by the operating system. If the hardware routinely runs out of free physical memory, more memory should be installed on the system immediately. Keep in mind buying more memory is relatively cheap but the crippling effects to a system without enough free memory can be very expensive. Internal testing has shown that if disk pages become necessary, an order of magnitude (1000%+) in slowdown is not uncommon.

How much RAM then is a good amount? This is going to depend upon many factors – for example: the applications and databases running on the platform, how many concurrent users, how much traffic is expected on the machine, etc. This said at least 2gigabytes per processor is a reasonable place to start – more for fewer processors, less for larger numbers of processors. In short, 12-16 gigs of RAM should meet most requirements as of this writing, and more memory should be used if running multiple jobs concurrently.

Disk I/O

If your flows use many IMPORT, EXPORT, SORT, ORAWRITE, TERAWRITE, DB2WRITE or TERAFEREAD operators, RETL will be using the disk a lot. In addition, if sorting data RETL will need to use temporary disk space in order to sort large datasets.

Some of the factors to consider when looking for fast I/O subsystems:

- **Access Speed** – the amount of time to access a location in a file is very important for RETL since RETL operations are usually heavily file I/O-intensive.
- **Overall Throughput** – places an upper bound on I/O performance for large data block movements. Important to RETL today – will become even more so as RETL matures.
- **Number of Independent Disks**– RETL is capable of using multiple disks via the TEMPDIR variable in the rfx.conf file. This can allow RETL to parallelize disk access and can have a significant impact on RETL performance.
- **RAID –** interleaving (striping) and mirroring data across disks can reduce I/O and improve performance. It is recommended to use hardware RAID whenever possible.
- **Mount options –** mounting a UFS filesystem using direct I/O can reduce I/O bottlenecks by eliminating the intermediate buffer in kernel address space. On Solaris, use the command mount with the '–o forcedirectio' option.

Architecture (e.g. 64bit vs 32bit)

Prefer 64-bit architectures.

64-bit architectures are supplanting 32-bit architectures. 64-bit architectures allow applications to address memory using 64-bits (much larger than the 2gigabyte limit imposed by 32-bit applications) and can often handle 64 bit mathematical operations and data transfers in one operation rather than multiple smaller operations required by 32-bit platforms.

If it is possible for any RETL flows to use more than 2 gigabytes of RAM, care should be taken so that customers use a 64bit machine and set any RETL parameters to use the 64bit JRE (11.x only, see the RETL Programmer's Guide for more details)

Processor Speed

Processor speed will determine the maximum theoretical limit on how many instructions can be processed per second – or in other words how fast data can be processed. Always prefer higher clock speed to more processors. One 800Mhz processor is going to be much faster than 8 100Mhz processors. How much faster is a good question and that is dependent upon many things. A rule of thumb might be to assume that there is a 30% overhead for 4-8 processors and even more for larger numbers of processors. As mentioned this is very dependent upon the O/S and many other factors.

Number of Processors

Can allow you to scale a particular hardware architecture cost effectively to deal with large problems. (see Processor Speed above).

The RETL framework can be optimized to maximize the use of CPUs. See the 'partitioning' sections of this guide and the Programmer's Guide.

Operating System Software

Different operating systems have different performance characteristics and will need to be configured differently to run RETL optimally.

AIX

Memory Consumption

Depending upon the flow and configuration settings RETL may need over 2 Gigs of RAM. If RETL is having trouble running – in particular if you notice that RETL is using around 2 Gigs of RAM – you should be checking this section.

ulimit

The amount of RAM a process can consume is specified in the /etc/security/limits file. However these values can be overridden via the ulimit command. We recommend that any environment running RETL has the following in the profile or .kshrc:

ulimit -m unlimited

This will override the limit specified in the /etc/security/limits file.

LDR_CNTRL

For versions of RETL 10.3.1 and earlier there is an issue that prevents the binary from allocating more than 2 Gigs of RAM on the 64 bit version of the executable (32 bit version of the executable is not capable of addressing 2Gigs of RAM). In short, there are some compiler options that allow us to get around this in RETL 10.3.2 and later version.

That said, you don't have to upgrade to fix this issue. There is an environment variable (LDR_CNTRL) to deal with this. This is the value (although probably not technically the best value) that we've been able to set it to get the desired result:

export LDR_CNTRL=MAXDATA=0xffffffffffffffffffff

SMP Tuning

Additionally, multi-processor AIX systems it seems have to be tuned on a per application basis for multithreaded applications if you'd like to take advantage of all of the CPUs, etc... Unfortunately, tuning these variables are dependent upon the number of processors involved, how many threads are running, etc. These are some values that we used to good success:

- export AIXTHREAD_SCOPE=P
- export AIXTHREAD_MNRATIO=3:1
- export MALLOCTYPE=buckets
- export MALLOCMULTIHEAP=1
- export AIXTHREAD_MUTEX_DEBUG=OFF
- export AIXTHREAD_COND_DEBUG=OFF
- export AIXTHREAD_RWLOCK_DEBUG=OFF
- export SPINLOOPTIME=1500
- export YIELDLOOPTIME=100

For more information try:

http://publibn.boulder.ibm.com/doc_link/en_US/a_doc_lib/aixbman/prftungd/2365c35.htm - HDRI45811

Solaris

Intimate Shared Memory

For RETL versions 11.0 and later, performance can be increased on systems with very large amounts of memory by using Intimate Shared Memory. See http://java.sun.com/docs/hotspot/ism.html

SMP Tuning

Solaris 8 introduced an alternate thread library that reduces the overhead in binding between user/application-level threads and kernel-level threads/lwps. To use the alternate thread libraries in Solaris 8, do the following :

```
export LD_LIBRARY_PATH=/usr/lib/lwp:${LD_LIBRARY_PATH}
```

The result should be better performance (Performance increases up to 20% have been seen in some cases), more scalability, and less overhead than the default Solaris 8 threads. This thread implementation is standard starting with Solaris 9. Again, this option should be used with caution and only on boxes with sufficient memory.

Version

It is not uncommon for OS's to be updated and patched. Check with Retek technical support to be sure that the version of RETL, the version of the DB and whatever other applications you intend to deploy are all supported by that version and patch level of the OS. Mismatches can cause performance problems or worse.

Architecture

Prefer 64 bit operating systems over 32 bit.

If you have a 64 bit hardware architecture it makes sense to also have a 64 bit operating system to take advantage of that great hardware. Without the 64 bit OS you cannot allocate more than 2 Gigs of RAM even though the hardware is capable of it. Therefore if you have 64 bit hardware use a 64 bit OS. Beware 64 bit operating system architectures can sometimes have 32 bit software components that can cause them to run even slower than native 32 bit architectures. This will get better over time and even now is not the norm.

The RETL bitset should match the hardware architecture bitset. In the 10.x versions of RETL, this means installing a different RETL binary. In the 11.x versions, this means setting JVM arguments to use different bitsets (see the Programmer's Guide for more information on this)

Scheduler

The scheduler affects the way that the OS handles processes and system level threads. Since RETL relies heavily on SMP this is a very important aspect of the OS. In AIX for example there are parameters that affect how the scheduler behaves (see AIX Performance Suggestions for more info).

Memory Management

How the OS deals with managing its memory. For example in AIX all threads allocate from a single heap thus serializing RETL to a large degree - unless some environment variables are set to change the default behavior (see AIX Performance Suggestions for more info).

File System

Not all file systems are created equally. File systems can be created to deal effectively with large files, or small files. Some file systems can automatically compress data coming in and decompress data going out. Some file systems perform journaling others do not.

It is important to choose the correct file system for RETL temp directories. Specifically you want a very fast file system that does NOT perform journaling, compression, etc. RETL needs to read and write many temporary (often large files). Never use NFS for RETL temp space – it is too slow.

Network

At this point in its lifetime, RETL is not a network program. Therefore we will limit our discussion within this domain. The only reason to talk about RETL and the network at the same time is for the following reasons:

Accessing a remote database – Co-locate where possible

RETL can be used to access a remote database – if used in this way be aware that network latency and throughput could have a drastic impact on RETL performance. The recommendation is to co-locate the RETL flow server and database server on the same box.

NFS

NFS is very slow - try to avoid using this at all costs. Note that this is very different than using specialized networks storage (e.g. one of EMC's solutions) which can be very very fast.

Moving RETL files

RETL input and output often needs to be moved before RETL can begin and/or when RETL is finished. These files are often large and cumbersome. Currently this is often done with FTP. The easiest thing you can do to improve performance for slow links (as well as decrease the probability of disconnects) is to compress (e.g. compress, gzip) the file before sending and uncompress (e.g. uncompress, gunzip) after receiving on the far side.

Database

Vendor (Oracle, DB2, Teradata)

Different databases have different strengths and weaknesses. They have different configuration parameters etc.

Version and Patch level of Database

Different patch levels and versions of the database have different performance and reliability characteristics. Be sure to contact support to make sure you are running on the correct version of the database for the product you are attempting to implement.

Indexes

Certain tables should be indexed to increase the speed of database queries and lookups. Remember that RETL uses SQL statements to load data just like other applications. Continue to tune and optimize your relations and tables as you would for any other application.

The general rule of thumb is to create appropriate indexes on all tables, including temp tables. This is very important, especially when doing nested selects and joins with these tables. Having an appropriate index can make an order of magnitude speedup in some queries.

Configuration

There are a wide variety of parameters that affect how the database performs. See the sections below specific to each database for more information.

For example, if setting up RDW, data warehouse templates and parameters should be used if RETL is going to be the primary point of access to the database.

Many multi-threaded applications that run on multi-cpu boxes become I/O-bound at some point. It is recommended to use mirrored RAID disks for any Oracle data files, or to specify a good distribution of disk usage when creating tablespaces. The idea is to load balance the I/O across as many disks as possible. See the references section regarding Tuning Disk I/O in Oracle

Hints

Some databases allow specification of ‘hints’ that tell the SQL query useful information the back end database engine can use to its advantage. For Oracle, one can use hints (See Oracle Hints in the references section) to give the Oracle optimizer ‘hints’ on how to parallelize SQL queries. Internal testing has shown a 10-20% performance improvement using these.

Logging

Create indexes/tables with NOLOGGING specified when possible, especially for temporary tables. This will notably speed up certain table operations that can minimize writing to the Oracle redo log. The syntax for doing this to an existing table is as follows: Alter table <TABLENAME> nologging;

RETL 11.2 and beyond allow specification of a 'createtableoptions' parameter where 'nologging' could be specified.

For database log files, ensure that separate log files are on independent disks or on RAID arrays with multiple disk controllers. For Oracle, ensure the redo log and undo tablespace are on separate disks from the RETL data. It is easy to hit an I/O bottleneck in redo log writing, particularly when using RETL to load data in conventional mode. It is recommended to have the redo log file in Oracle on mirrored RAID disks.

Minimize the REDO log disk activity. It is recommended to specify larger-than-default values of log_buffer, log_checkpoint_interval, log_checkpoint_timeout, etc so that fewer commit points to the redo log can minimize disk I/O activity, or better yet specify NOLOGGING as above.

Logic in the Database vs Logic in RETL

Generally each technology should be maximized where it performs best. Database technology has been around for decades and is optimized with indexes and the like. As a result, much functionality should be moved into the database when it makes sense from a performance and overall maintainability standpoint.

For example, generally it is better to move sorting functionality into the database when a database table has an usable and valid index on it. This can be done either by having an ORDER BY clause in the query, or by having the table index set up so that inserts to the table will automatically be in sorted order.

Allowing RETL to parallelize database queries

Beginning with RETL 11.2, it is possible to specify multiple QUERY properties to facilitate with RETL partitioning. To take advantage of this, select statements should be broken into multiple queries, or tables should be partitioned so that select statements can be one per partition. Take the following flow:

```
<FLOW>
   <OPERATOR type="oraread">
   <PROPERTY name="query" >
      <![CDATA[
         Select * from table where columnA = 1
            OR columnA = 2
            OR columnA > 3
      ]]>
   </PROPERTY>
…
</FLOW>
```

This flow can be split into three queries, one for each case in the select statement:

```
<FLOW>
   <OPERATOR type=”oraread”>
   <PROPERTY name=”query” >
      <![CDATA[
         Select * from table where columnA = 1
      ]]>
   </PROPERTY>
   <PROPERTY name=”query” >
      <![CDATA[
         Select * from table where columnA = 2
      ]]>
   </PROPERTY>
…
</FLOW>
```

Data

How it is partitioned

Generally, it is better if data is broken into pieces so it can be processed in parallel. This provides for load balancing among worker threads independently working on the same task. Partitioning can be done at the database level or by breaking data into separate files. The effect is to help remove I/O as a bottleneck and increase parallelism in a single process. Partitioning is generally a good thing for RETL – but it can result in increased maintenance costs so should only be done if required. RETL uses a hash-based (hash operator), round robin (splitter operator), import, and database read partitioning scheme at the time of this writing.

Hash-based data partitioning via the HASH operator

There are issues with hash-based partitioning in that it is heavily reliant on the data not being skewed towards one partition. If the data being hashed on a set of keys is heavily skewed towards one partition, there will be an unbalanced load on that partition, and the effects of this are either less-than-expected speedups in performance or—at worst—degradations in performance due to additional overhead in the data split. The main idea here is that to make hash-based partitioning work in RETL, the hash keys that the data is partitioned by should allow the data to be uniformly distributed among partitions.

The general rule of thumb is to partition data based on a sequence key to ensure equal distribution among partitions. This will allow RETL to effectively load balance among partitions most effectively. However, many RETL operators require partitioning based on other keys in order to JOIN, GROUPBY, CHANGECAPTURE, etc. Care should be taken to determine what effect the data skew will have on partitioning in these cases.

Round robin data partitioning via the SPLITTER operator

Generally, most flows will require hash-based partitioning if there are any operators that require sorted input (e.g. JOIN variants, CHANGECAPTURE, GROUPBY, etc). However if some flows don't contain operators that require sorted input (e.g. LOOKUP, BINOP, FIELDMOD, CONVERT, etc), round robin partitioning will work the best. Round robin partitioning ensures that data is evenly split up among partitions and that data skew to one partition is kept to an absolute minimum. Generally, flows that contain SPLITTER as a partitioning operator will scale better than those that use HASH as a partitioning operator.

Import data partitioning via the IMPORT operator

Like round robin data partitioning, import partitioning works if the flow does not contain operators that require sorted input. If working with one input file, import partitioning avoids data skew to one partition by processing roughly equal subsets of the file in each partition. When one file is specified for each partition, data skew is dependent on the input files.

Database read partitioning via DBREAD, DB2READ, and ORAREAD

Like round robin and import partitioning, database read partitioning works if the flow does not contain operators that require sorted input. Because the data being handled by each partition is determined by the specified queries, queries should be constructed such that data skew towards one partition does not overcome the benefits of partitioning.

Where it is located

Generally it is best to make sure that RETL is co-located with the data it is manipulating. This does away with network latency and other complications that can cause problems. Additionally, when dealing with flat files it makes sense to use multiple disks – files can be read off of more than one disk at a time – increasing the I/O bandwidth.

Don't run RETL with data files that are located on network or NFS-mapped drives.

How it is normalized in the database

This is the relationship between database tables and data elements. In order to be more space efficient, database architects will often create relationship tables between common data elements and you need to do an additional table lookup or join to get additional information about an element, which slows performance. These types of tradeoffs can affect the bottom line performance of flows by removing the need to do joins and/or lookups, etc.

Number of fields per record

The more data, the longer it takes to run a flow. Filtering out to the bare number of fields required for the flow will allow the flow to run more quickly.

Field data types

String fields are faster than the other data types when read from or written to a file (IMPORT and EXPORT) because there is no need to translate the characters in the file to an internal type such as an integer or a floating-point number, and vice versa. If a field is not involved in any numerical calculations or comparisons, consider making the field type string. This may not apply to numerical fields read from a database because this is just pushing the conversion onto the database instead of being performed in RETL.

Number of records

RETL was built to take advantage of multiple processors and additional memory, however more records will mean more time to run flows. RETL was built to be able to scale, but remember that certain operators do NOT scale linearly. Specifically sort is n log n and join is n^2 worst case – therefore when these operators involved doubling the number of records may cause the flow to take MORE than twice as much time.

Sorting

Sorted data can be problematic for proper flow control and design. If you satisfy the flow requirements without sorted data, then do so. If you do need to have sorted data in your flow, the following tips can potentially help you improve the performance of your flows.

- If you are getting your data from a database, consider having the database sort the data (using the ORDER BY clause in the query) instead of RETL. The database may be able to use indexes to sort the data significantly faster than RETL.

  Internally, RETL does not recognize that data coming from the database to be sorted, even if it is. Some operators will display warnings about requiring sorted data to work correctly. These can be safely ignored if the data is sorted by the database.

  Moving the sort to the database will put an additional load on the database, so this may not be a good option if the database is already overworked.

- If an IMPORT is immediately followed by a SORT, RETL first reads the file, then writes the file back to disk for sorting, then re-reads in the sorted data. Consider sorting the data before running RETL. The gsort utility is distributed with RETL and can be used in this circumstance. If you do pre-sort the input file, remember to change the schema file to indicate that the data is sorted. This will eliminate warnings displayed by some operators requiring sorted data.

  One caveat to pre-sorting the data is that using multiple threads to input a file (by setting the "degreesparallel" property), will probably not provide a performance benefit.

- If you have multiple sorts in a flow, look for ways to combine the sorts. For example, suppose you have a sort on field "A" then later in the flow there is a sort on fields "A" and "B". It may be possible to perform the first sort on both fields and eliminate the second sort. This may require reorganizing the flow. It's best to work from finest grained sort to coarse-grained sort because some operators may manipulate the data and put the finer-grained sort out of order.

- The "numsort" property of SORT can be specified to have RETL perform the sort using multiple threads. Each thread will sort a portion of the records, with the results sort-funneled back together. This will most likely improve performance on multiple-CPU servers. See the Programmer's Guide section on the SORT operator for more information. The best performance will probably be obtained when the number of threads is equal to the number of CPUs.

- Consider using data partitioning. Data partitioning requires some effort to get flows working correctly, but can result in significant performance increases.

- If it is not possible to pre-sort imported flat files, a good way to reap the benefits of data partitioning without analyzing the whole flow is to use IMPORT→SORT→SORTFUNNEL and specifying a value for the "numpartitions" property in the IMPORT.

RETL-based factors

Flow design

The design of the flow can have an enormous impact on the performance of the flow. Here are several suggestions for making a flow perform better:

- Focus on relevant data EARLY in the flow. In short, get rid of data that you don't need by using FILTER, FIELDMOD or whatever other means comes to mind.
- Try to avoid sorting and joining data that you don't need – these operations are expensive. If normal performance is lacking with a sort operator, try moving this operation to the database if possible. This can be achieved by using an "ORDER BY" statement in the SQL query.
- Avoid using hash->sort operations unless partitioning is set to be greater than 1 and has been shown to increase performance for that flow.
- Avoid the use of the SORTFUNNEL operator. This operator may increase the likelihood of deadlock in 'circular' flows, which may result in paging to disk. Disk paging is extremely expensive and as such it is recommended to avoid the SORTFUNNEL operator.
- Minimize usage of the FUNNEL operator. Oftentimes a FUNNEL operator will prevent a flow from maximizing partitioning and can effectively serialize processing in that section of the flow.
- Minimize the use of operators that require sorted input where possible, instead substituting an operator that doesn't require sorted input. For example, LOOKUP can be substituted for INNERJOIN. Operators that require sorted input reduce the degree of parallelism that the RETL framework is able to achieve, namely in IMPORT and ORAREAD operators.
- Use the LOOKUP operator to join when using a relatively small lookup table. Since LOOKUP will hold records on the lookup dataset in memory, LOOKUP can perform very quickly on small datasets. However, if the lookup table is large, performance may be negatively affected. In general, if the host system has enough capacity for the highest volume to be expected on the lookup dataset, LOOKUP should perform well. However, if there is little free memory for LOOKUP or there is an arbitrarily large data volume expected on the lookup dataset, INNERJOIN or LEFTOUTERJOIN is recommended instead of LOOKUP.

Buffer Size

The bufsize parameter from your rfx.conf file affects the size of buffers that connect operators. Large values will mean that larger blocks of records will be accumulated before passing to the next operator. This will mean less synchronization between operators and potentially higher performance – however, making the values too high will degrade performance. Though the default value (2048 as of this writing) is a reasonable value - this value will need to be tuned to fit your local environment and your flows. Tuning this value may result in performance gains/loss of +/- 20%.

Number of Partitions

The number of partitions (i.e. numpartitions in rfx.conf) determines how aggressively RETL will use threading. The value has a multiplicative effect for some flows (e.g. with the HASH operator) – multiplying the number of threads that a flow consumes.

A value of 1 is the most conservative value and generally the one recommended. Increasing this value will cause RETL to break up data, try to parallelize it and then try to join the data back together by the end of the flow. Since the RETL tends to use threads aggressively – this value can often cause system level thrashing between threads and can actually degrade performance.

Increase this value (slowly) with caution, making sure to test for performance with each change.

You should specify as many TEMPDIRS on independent disks as # of partitions. So, for example, if you have numpartitions set to 4, you should ensure that you have 4 TEMPDIRS on separate physical disks.

For a detailed write-up on partitioning, refer to the Programmer's Guide.

Number of Operators

The fewer operators in a flow the faster it will generally run. The exception to this rule is for those operators that narrow the focus of the flow and get rid of unneeded records (e.g. FIELDMOD and FILTER). In general, the flow developer should try to minimize the number of operators in the flow.

10.x versions of RETL -- Each operator consumes at least one thread – thus an 80 operator flow will have 80 or more threads as of this writing.

11.x versions of RETL – 11.x versions of RETL are more conservative in their use of threads. Each operator may consume a thread, but certain operations have been placed in single-threaded pipelines to minimize extra threads and any synchronization that is necessary among threads.

Type of Operators

Certain operators generally take longer to perform their tasks than others. The following operators are among the most expensive in terms of performance (in no specific order): import, sort, join, groupby, dbread, dbwrite. Many of these operators also happen to be very fundamental and are often necessary when processing data with RETL. These are the types of operations that should be scrutinized and tuned when performance problems have been identified.

See the "Operator Tuning" section for more information on specifics of tuning each operator.

See the "Flow design" section for more information on other operators and how they affect performance.

Operator Tuning

Operators often contain a number of properties that can drastically aid in improving a flow's overall performance.

Operator	Property	How it affects performance
ORAREAD	arraysize	The 'arraysize' property specifies the number of rows for the Oracle driver to pre-fetch. This allows the database reader to 'fetch' multiple rows with each trip to the database. This reduces the number of trips that RETL will need to make to the database. Increasing this value will increase performance to a point; however further increases may offer diminishing to negative returns. Additionally, increasing the value of 'arraysize' increases the amount of memory required. This property should be adjusted in a localized fashion instead of a global fashion.
	N/A	ORAWRITE uses SQL*Loader as the underlying load mechanism to Oracle databases. As a result, many of the performance tuning features available for tuning SQL*Loader are transitively tunable in RETL. Please see the Oracle 9i Database Utilities->SQL*Loader guide for a more complete reference of SQL*Loader. A full explanation of all the features of SQL*Loader is beyond the scope of this document.

Operator	Property	How it affects performance
	method	The 'method' property can make drastic changes to the performance characteristics of flows that contain ORAWRITE operators. The two valid values for this property are the following: **Conventional** – SQL*Loader creates INSERT statements to load into the database. This is the slowest but generally the most reliable and constraint-free method of loading to an Oracle table. When in doubt, conventional mode should be used unless performance dictates otherwise. **Direct** – SQL*Loader bypasses many of the intermediate buffers and integrity checks required when loading via conventional mode. As a result, it can be several times faster to perform loads via the direct method of SQL*Loader. However, this method should be used with care, as it comes with additional strings attached (e.g. indexes on the table need to be rebuilt after a direct load, and integrity constraints are disabled until the load completes). The Oracle documentation should be consulted for further information regarding direct mode.
	numloader	The 'numloader' option is a great way to improve and size performance of flows with ORAWRITE that run on boxes with many CPUs and many independent disks. This specifies the number of SQL*Loaders that will simultaneously load into the database. The number of degrees of parallelism should map to the number of independent disks (ideally on multiple controllers) listed as TEMPDIRS in rfx.conf in order to minimize I/O contention and maximize disk throughput.  **Note:** This should usually only be used on flows with partitioning turned off (e.g. numpartitions=0 in rfx.conf)

Operator	Property	How it affects performance
	parallel	The 'parallel' property is used along with partitioning to specify that ORAWRITE should write to the database in parallel. This can provide for drastic performance gains in flows that use partitioning.
	mode	The 'mode' property specifies the loading operation performed on the table. This property should be set to 'truncate' when replacing rows in an existing table to achieve best possible performance.
	outputdelimiter	The 'outputdelimiter' property can be used to specify SQL*Loader to run in delimited format. By default, ORAWRITE writes a SQL*Loader datafile in fixed mode, which is the best-performing mode in most cases. However, delimited format may be faster in certain cases, namely when there are fields that may have large amounts of trailing spaces, or 'padding'. If there are these such sparsely populated data columns in the incoming data, using delimited mode in orawrite has shown performance to increase up to 30%.

Operator	Property	How it affects performance
ORAWRITE	loaderoptions	Starting with RETL 11.2, any SQL*Loader options can be specified and passed on through RETL via the 'loaderoptions' property. A few of the options that have been promising: **ROWS** -- When specified with a conventional load, it represents the number of rows per commit. This can drastically increase performance of conventional loads since the default number of rows per commit is 64 rows. For direct loads, this setting specifies the number of save points before the load is complete. The default for direct load is to load all rows before saving; as a result, it is not recommended to specify the rows property when operating loading via the direct method. **SORTEDINDEXES** -- Can specify if and how data is sorted when it reaches SQL*Loader. This can drastically speed up loads when the incoming data to SQL*Loader is sorted by the same keys as the table's index. This is only allowed in direct path loads. **SINGLEROW** -- Can be specified in direct path loads when appending to existing tables. This can improve performance when appending a small number of records into a very large table. Internal tests using RETL has shown performance increases up to 50% faster when appending a small number of rows to a very large table. However, in most cases, it should be avoided since it is likely to slow down performance.
	query	Specify multiple queries to increase the degree of parallelism when using partitioning in RETL, by allowing concurrent query execution. See the Programmer's Guide for more information

Operator	Property	How it affects performance
IMPORT	numpartitions	Specifies the number of partitions import should split the input data file into. Increasing this value may increase the degree of parallelism in the flow by allowing RETL to parallelize file reads. See the Programmer's Guide for more information
DB2WRITE, DBWRITE, ORAWRITE, EXPORT, GROUPBY, CLIPROWS, DEBUG	parallel	Set to 'true' to allow enabling of parallelism in each operator. For specifics on how 'parallel' property affects each operator, please refer to the Programmer's Guide.
SORT	numsort	To try to increase the performance of some flows, it may become necessary to enable multi-threaded sorting. This allows for improved sort performance by splitting the data into n number of threads, sorting each thread individually and then applying a sortfunnel on the resulting threads to bring the records back together in sorted order. To utilize multi-threaded sorting, set the "numsort" property. Optimal performance will probably be obtained when "numsort" is set to the number of processors the machine has available for sorting.
	N/A	Another aspect of multi-threaded sorting that should be examined is temporary files/directories. RETL accepts multiple directories for temporary files. SORT accesses these directories in a round-robin fashion. It is preferable to have a separate temporary directory for each thread for better I/O performance. If you provide 2 temporary directories to RETL and specify numsort=3 in a SORT, then the first tmp directory would be used by the first and third threads, while the second thread will use the second tmp directory. This would potentially cause excessive disk I/O in the first tmp directory.

Version of RETL

RETL is still a young product and new patches and versions can often have significant leaps forward in terms of performance. As always – if it works for you don't upgrade! However checking with support to see if there are any performance related fixes/patches may be one of the fastest, easiest, and least-costly ways to improve the performance of your flows.

Java Virtual Machine Settings (RETL 11.x only)

RETL was re-written on a java-based architecture starting with RETL 11.0. As a result, RETL 11 runs within a Java Virtual Machine (JVM) rather than in native code like the 10.x versions. Given that RETL is typically used on large boxes with larger physical memories and many CPU's, an aggressive stance towards JVM resource management is taken. That is to say, in most cases, RETL will attempt to use more rather than fewer resources.

This document will not go into details regarding the JVM internals, but will recommend some settings that can be made so that RETL can be more performant. The JVM flags allow users to do things such as tweaking memory usage, tuning object recycling/garbage collection, and modifying compilation/startup time. As always, any changes here should only be put into effect when 1) a performance problem has been identified, and 2) performance has been positively affected by the change.

Note: Flags can be passed on to the JVM through the RETL environment variable JAVA_ARGS or via various environment variables. Please run RETL with the –e option to get more information about these various environment variables

There are certain flags that can help in diagnosing time spent in garbage collection. A brief list of these is as follows:

Garbage Collection statistics and profiling

The garbage collection activity can be monitored by doing the following:

export JAVA_ARGS=" -verbose:gc ${JAVA_ARGS}"

and

export JAVA_ARGS=" –XX:+PrintGCTimeStamps ${JAVA_ARGS}"

and

export JAVA_ARGS=" –XX:+PrintGCDetails ${JAVA_ARGS}"

Additionally, a tool available from Sun (For the Solaris JVM) called GC Portal can be handy for sizing out heap sizes.

RETL_VM_MODE

The mode in which to run RETL – by default RETL runs in high volume mode. This should be the default in most if not all cases. However, if startup time is critical, one can run RETL in low volume mode by setting the RETL_VM_MODE environment variable to 'lowvol'. In general, only flows that are desired to run in a few to several seconds should use the 'lowvol' mode. All other flows should not set this environment variable.

JVM Heap Sizing

The JVM memory heap size can be sized so that the JVM will minimize the time spent in garbage collection, which will increase the overall throughput of RETL. Tuning the JVM heap size has been shown to positively increase the performance of RETL by 50% or more in some cases.

The simplest way to tune the JVM is to set the JVM heap size so that it is as large as possible for the host environment. RETL_INIT_HEAP_SIZE and RETL_MAX_HEAP_SIZE correspond to the JVM parameters –Xms and –Xmx, which affect the minimum and maximum heap sizes respectively. Since JVM heap resizing can be expensive and resizing is often accompanied by a full garbage collection, changing the minimum heap size so that it is large enough to prevent heap re-sizing can increase overall throughput. Ideally, RETL_INIT_HEAP_SIZE should be equal to RETL_MAX_HEAP_SIZE so the JVM heap doesn't need to serially resize the heap (often preceded by a full gc) in a 'stop-the-world' operation. This all being said, the heap size should not exceed the maximum amount of physical memory. Also, it should not reserve too much unnecessary heap space so that other processes that may need to run concurrently may do so efficiently. Not following these guidelines will make the O/S resort to disk paging, which will be counterproductive to having large heap sizes.

A more automatic way of tuning the heap (and garbage collector, for that matter) can be accomplished by setting JAVA_ARGS to '-XX:+AggressiveHeap', which tells the JVM to be 'aggressive' about its memory usage and garbage collection strategy. Again, this option should be used with care and only on RETL flows that run serially and/or that need the high performance characteristics that an aggressive heap can provide.

Note: when running RETL with large heaps (> 2GB), the 64bit JVM will need to be used. This is usually accomplished by setting RETL_ENABLE_64BIT_JVM=1 and RETL_MAX_HEAP_SIZE to the appropriate settings. It is recommended to not use the 64bit JVM unless a RETL flow can make use of a very large heap size. Even so, the 64bit JVM may still run quite a bit slower than a 32bit JVM of the same version. In internal tests, the 64bit JVM can run 10-15% slower than it's corresponding 32bit JVM, if a RETL flow cannot take advantage of a large heap.

JRE Version

RETL comes shipped with a JVM (RETL 11.1 contains the 1.4.1 JVM). Significant performance improvements can be gained simply by upgrading the JVM in some cases. The new location of the newer Java Runtime Environment can be specified in the environment variable RETL_JAVA_HOME.

Garbage Collection Algorithm

There are multitudes of ways to configure the garbage collector in use by RETL. It is beyond the scope of this document to go into every detail here, but there may be certain garbage collection algorithms with certain heap size settings that could make performance better.

Since RETL is usually run on SMP machines with large numbers of CPUs, we recommend running RETL with the Throughput Collector. The Throughput Collector runs multiple memory allocation and garbage collection threads in parallel, which significantly speeds up the overall throughput for RETL. To turn this option on, do the following:

```
export JAVA_ARGS=" -XX:+UseParallelGC ${JAVA_ARGS}"
```

Note: This will work on Solaris only and is set by default in RETL 11.2 versions and above

There is another JVM option that will allow the garbage collector to be more intelligent about how it sizes the new and old generations of the heap. The Adaptive Size policy has been shown to improve performance by 20% or more on certain flows. To turn this option on, do the following:

```
export JAVA_ARGS=" -XX:+UseAdaptiveSizePolicy ${JAVA_ARGS}"
```

Note: This will work on Solaris only and is set by default in RETL 11.2 versions and above

Other Settings

If any users experiment with any other JVM options and find optimized settings for their flows and environment, the RETL development team would love to hear about them. Please email Josh.Chu@Retek.com if you have found that other non-default garbage collectors have positively influenced your performance.

References

displaying debugging and performance tuning information with tusc

http://h21007.www2.hp.com/dspp/tech/tech_TechDocumentDetailPage_IDX/1,1701,2894,00.html

Common Performance Bottlenecks And How to Find Them

http://www.colltech.com/real_world_solutions/bottlenecks.html

IBM's System Performance Tuning Guide (for AIX)

http://www.redbooks.ibm.com/redbooks/SG245340.html

Performance Management Guide

- General Performance Notes and Specific Information for AIX.

http://publibn.boulder.ibm.com/doc_link/en_US/a_doc_lib/aixbman/prftungd/prftungd02.htm

AIX Versions 3.2 and 4 Performance Tuning Guide

http://nscp.upenn.edu/aix4.3html/aixbman/prftungd/toc.htm

SUN PERFORMANCE

http://www.sun.com/sun-on-net/performance/

Alternate thread library on Solaris 8

http://developers.sun.com/solaris/articles/alt_thread_lib.html

Performance Engineering: A Practical Approach To Performance Improvement

http://www.rational.com/products/whitepapers/307.jsp

Building a database for performance (free login required)

http://download-west.oracle.com/docs/cd/B10501_01/server.920/a96533/build_db.htm

Oracle 9i database utilities

http://download-west.oracle.com/docs/cd/B10501_01/server.920/a96652/toc.htm

Oracle Hints

http://www.oradev.com/hints.htm

Turbo-charging the Java HotSpot Virtual Machine, v1.4.x to Improve the Performance and Scalability of Application Servers

http://developer.java.sun.com/developer/technicalArticles/Programming/turbo/

The Java HotSpot Virtual Machine

http://java.sun.com/products/hotspot/docs/whitepaper/Java_HotSpot_WP_Final_4_30_01.html

Big Heaps and Intimate Shared Memory

http://java.sun.com/docs/hotspot/ism.html

Java 1.4.1 command-line syntax on Solaris

http://java.sun.com/j2se/1.4.1/docs/tooldocs/solaris/java.html

Tuning Disk I/O in Oracle 8i

http://www.oracle.com/oramag/oracle/99-Nov/index.html?69tun.html

Tuning the Java Runtime for "Big Iron"

http://java.sun.com/developer/community/chat/JavaLive/2001/jl0327.html

Tuning Garbage Collection with the 1.4.2 Java Virtual Machine

http://java.sun.com/docs/hotspot/gc1.4.2/index.html