TIBCO ActiveSpaces® Transactions

System Sizing Guide

Software Release 2.5.8
Published November 10, 2017

Two-Second Advantage®

Important Information

Contents

List of Figures

List of Examples

About this book

This guide describes how to size systems for deploying TIBCO ActiveSpaces® Transactions applications. It provides a description of the parameters that need to be estimated and a template for approaching configuring a system correctly for optimal performance.

This book is intended for the following types of readers:

- System administrators planning for deploying a TIBCO ActiveSpaces® Transactions application.
- TIBCO ActiveSpaces® Transactions developers providing recommendations on hardware requirements for a TIBCO ActiveSpaces® Transactions application.

This guide is organized into these general areas:

- Description of the required metrics to estimate the system requirements for a TIBCO ActiveSpaces® Transactions application. This provides an overview of the metrics that must be measured or estimated to accurately estimate the hardware requirements for a TIBCO ActiveSpaces® Transactions application. This information is in Chapter 2.
- A step by step approach to estimating the hardware requirements for a TIBCO ActiveSpaces® Transactions application. This chapter provides a template that can be used to determine the system requirements. This information is in Chapter 3.

This book is part of a set of TIBCO ActiveSpaces® Transactions documentation, which also includes:

TIBCO ActiveSpaces® Transactions Installation — This guide describes how to install the TIBCO ActiveSpaces® Transactions software.

TIBCO ActiveSpaces® Transactions Quick Start — This guide describes how to quickly get started using Java IDEs to develop TIBCO ActiveSpaces® Transactions applications.

TIBCO ActiveSpaces® Transactions Architect's Guide — This guide provides a technical overview of TIBCO ActiveSpaces® Transactions .

TIBCO ActiveSpaces® Transactions Administration — This guide describes how to install, configure, and monitor a TIBCO ActiveSpaces® Transactions deployment.

TIBCO ActiveSpaces® Transactions Java Developer's Guide — This guide describes how to program a TIBCO ActiveSpaces® Transactions application.

TIBCO ActiveSpaces® Transactions Performance Tuning Guide — This guide describes how to performance tune TIBCO ActiveSpaces® Transactions applications.

TIBCO ActiveSpaces® Transactions Javadoc — The reference documentation for all TIBCO ActiveSpaces® Transactions APIs.

Conventions

The following conventions are used in this book:

Bold — Used to refer to particular items on a user interface such as the **Event Monitor** button.

`Constant Width` — Used for anything that you would type literally such as keywords, data types, parameter names, etc.

`Constant Width Italic` — Used as a place holder for values that you should replace with an actual value.

TIBCO ActiveSpaces® Transactions community

The TIBCO ActiveSpaces® Transactions online community is located at `https://devzone.tibco.com`. The online community provides direct access to other TIBCO ActiveSpaces® Transactions users and the TIBCO ActiveSpaces® Transactions development team. Please join us online for current discussions on TIBCO ActiveSpaces® Transactions development and the latest information on bug fixes and new releases.

1

Introduction

What is TIBCO ActiveSpaces® Transactions ?

TIBCO ActiveSpaces® Transactions is an in-memory transactional application server that provides scalable high-performance transaction processing with durable object management and replication. TIBCO ActiveSpaces® Transactions allows organizations to develop highly available, distributed, transactional applications using the standard Java POJO programming model.

TIBCO ActiveSpaces® Transactions provides these capabilities:

- Transactions - high performance, distributed "All-or-None" ACID work.
- In-Memory Durable Object Store - ultra low-latency transactional persistence.
- Transactional High Availability - transparent memory-to-memory replication with instant fail-over and fail-back.
- Distributed Computing - location transparent objects and method invocation allowing transparent horizontal scaling.
- Integrated Hotspot JVM - tightly integrated Java execution environment allowing transparent low latency feature execution.

System requirements

To optimize the performance of TIBCO ActiveSpaces® Transactions applications the appropriate system resources must be allocated. The system resources that must be considered are:

- System memory

- Number and clock speed of processing units
- Disks
- Network speed and interfaces
- High availability and the number of machines

System memory

The total amount of physical memory in a system must be determined. TIBCO ActiveSpaces® Transactions applications derive a large percentage of their performance from caching as much as possible in memory. TIBCO ActiveSpaces® Transactions applications use both shared memory to cache Managed Objects and JVM heap space for allocation of non-Managed Java objects.

Processors

The TIBCO ActiveSpaces® Transactions transactional environment provides scaling across multiple processing units in a single system. The total number of processors in a system directly impacts the total through-put of a TIBCO ActiveSpaces® Transactions application. The clock speed of the processing units has a direct impact on the cost of a single execution path in an application. The faster the clock speed, the lower the latency when executing non-contested application logic.

Disk

In general, TIBCO ActiveSpaces® Transactions applications cache application data in shared memory. However, the size and number of disks must be determined to accommodate any application specific logging that is required.

Network speed

The total aggregate message through-put of a TIBCO ActiveSpaces® Transactions application is impacted by the network bandwidth. The network bandwidth must be large enough to not limit the number of requests and responses that can be processed by a TIBCO ActiveSpaces® Transactions application and to support the communication for Distributed, Mirrored and Replicated objects.

Number of machines

Multiple machines are allocated to TIBCO ActiveSpaces® Transactions applications to support through-put scaling and also redundancy for highly available applications.

2

Sizing Metrics

This chapter describes the metrics that must be measured or estimated to determine the system requirements for a TIBCO ActiveSpaces® Transactions application.

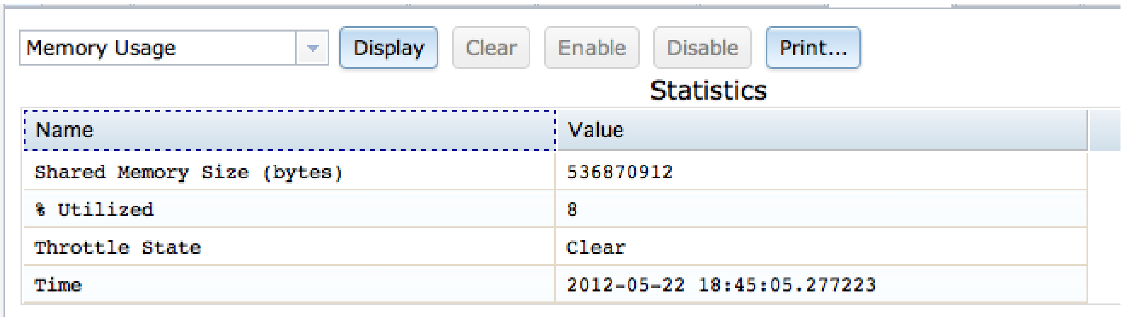
Memory

Shared memory

Shared memory usage has these parts:

- a base system usage by the TIBCO ActiveSpaces® Transactions runtime
- usage by Managed objects
- in-flight transaction logging
- usage by replica objects
- usage by cached objects

The base system usage for an TIBCO ActiveSpaces® Transactions node can be determined by displaying the `Memory Usage` from the node `Statistics` tab on an unloaded node (i.e. No application running, no application usage of memory). This display reports on the total shared memory, the current shared memory usage and the current shared memory throttling state:



Name	Value
Shared Memory Size (bytes)	536870912
% Utilized	8
Throttle State	Clear
Time	2012-05-22 18:45:05.277223

Figure 2.1. Memory Usage

As shown in Figure 2.1, an TIBCO ActiveSpaces® Transactions node base system usage is 21% of 512 megabytes, or 100 megabytes. This base footprint will increase slightly as managed object classes are loaded.

Managed objects persist their state in shared memory until they are explicitly deleted by the application. The shared memory usage of a managed object can be determined programmatically by creating an instance of the managed object, populating it with typical data values, and passing it to the `com.kabira.platform.osstats.Type.memorySize()`method

The call to the `Type.memorySize()` method must be done in a separate transaction from the object creation to get accurate results.

This is shown in Example 2.1 on page 4:

Example 2.1. Object size snippet

```
// $Revision: 1.1.2.4 $
package com.kabira.snippets.sizing;

import com.kabira.platform.ManagedObject;
import com.kabira.platform.Transaction;
import com.kabira.platform.annotation.Key;
import com.kabira.platform.annotation.Managed;
import com.kabira.platform.osstats.Type;

/**
 *  Display Managed object memory usage.
 * <p>
 * <h2> Target Nodes</h2>
 * <ul>
 * <li> <b>domainnode</b> = A
 * </ul>
 */
public class ObjectSize
{
    /**
     * Sample application object
     */
    @Managed
    @Key(name = "ByName", fields =
    {
        "name"
    }, unique = true, ordered = false)
    private static class MyObject
    {
        static final int NUMBER_OF_ELEMENTS = 100;
        String[] stringArray;
        final String name;

        /**
         * When created, populate this instance with some data
         */
        public MyObject(String name)
        {
            this.name = name;

            stringArray = new String[NUMBER_OF_ELEMENTS];

            for (int i = 0; i < NUMBER_OF_ELEMENTS; i++)
            {
                stringArray[i] = Integer.toString(i);
            }
        }
    }

    /**
     * Main entry point
     * @param args  Not used
     */
    public static void main(String[] args)
    {
        //
        // Create the objecct
        //
        new Transaction("Create Object")
        {
            @Override
            protected void run() throws Rollback
            {
                MyObject myObject = new MyObject("Sample");
            }
        }.execute();

        //
        // Report object size - this must be done in a separate
        // transaction.  It only works for committed objects.
        //
        new Transaction("Report Object Size")
        {
            @Override
            protected void run() throws Transaction.Rollback
            {
                for (MyObject myObject : ManagedObject.extent(MyObject.class))
                {
                    System.out.println(new Type().memoryUsage(myObject));
                    ManagedObject.delete(myObject);
                }
            }
        }.execute();
    }
}
```

This program's output will be similar to the following:

```
Allocation type: # of bytes, allocator bucket size, notes
=====================================================================
metadata: 480, 592, spaces: [allocation=64] [system=24] [lock=112]
key com.kabira.snippets.sizing.ObjectSize$MyObject::ByName: 128, 208
array com.kabira.snippets.sizing.ObjectSize$MyObject::stringArray: 1006 (aligned 1008),
 1168
optimal allocationSpaceBytes = 1032
event queue: 0, 0
Total: 1614 1968
```

The output has three columns:

1. `Allocation type` - what part of the object this allocation deals with.
    - `metadata` - Object data. This allocation is broken into two or three parts. The `system` space which includes the storage for all fixed length object fields, and pointers to non-fixed length fields, and the runtime overhead. The `allocation` space is the size of the allocation done to store the data for variable length object fields.

      For types where the `dynamicLockMemory` element of the `@Managed` annotation has been left (or set to) its default setting of `false` there will also be a lock portion of the metadata, which shows the size of the space being used for the transaction lock.
    - `transaction lock memory` - For types where the `dynamicLockMemory` has been set to `true` there will be an additional line of report output, which shows the space used by the transaction lock that is dynamically allocated and de-allocated each time the object is locked within a transaction:

      ```
      transaction lock memory: 112, 208, dynamic
      ```

      This setting reduces memory utilization by instances of objects when they are not locked within a transaction at the cost of increased CPU path length and reduced scalability when the object is accessed.
    - `key` - One entry for the value of each defined key. This allocation is used to populate the indexes and is a separate allocation from the storage for the object fields covered by the key.
    - `array` - An array field, including storage for the array elements.
    - `string` - A string field, including storage for the data.
2. `# of bytes` - the number of bytes requested for this part of the allocation.
3. `allocator bucket size` - the number of bytes actually allocated. Allocations are fitted to the nearest fixed shared memory allocator bucket size.

The final line of the report shows the total memory requested and the actual memory allocated.

An addition line of the report shows the optimal size for the `allocation` space. At object creation, by default, the runtime chooses a small value for the `allocation` space. User data for strings and arrays that does not fit within this space causes additional memory allocations. If the optmial size is different than the `allocation` space shown in the `metadata` line of the report space and peformance savings may be gained by expliciting setting it via the `allocationSpaceBytes` element of the `Managed` annotation. From the example above, change the setting to 1032 bytes:

```
    @Managed(allocationSpaceBytes = 1032)
    @Key(name = "ByName", fields =
    {
       "name"
    }, unique = true, ordered = false)
    private static class MyObject
```

Re-running the modified class, results in:

```
    Allocation type: # of bytes, allocator bucket size, notes
    =====================================================================
```

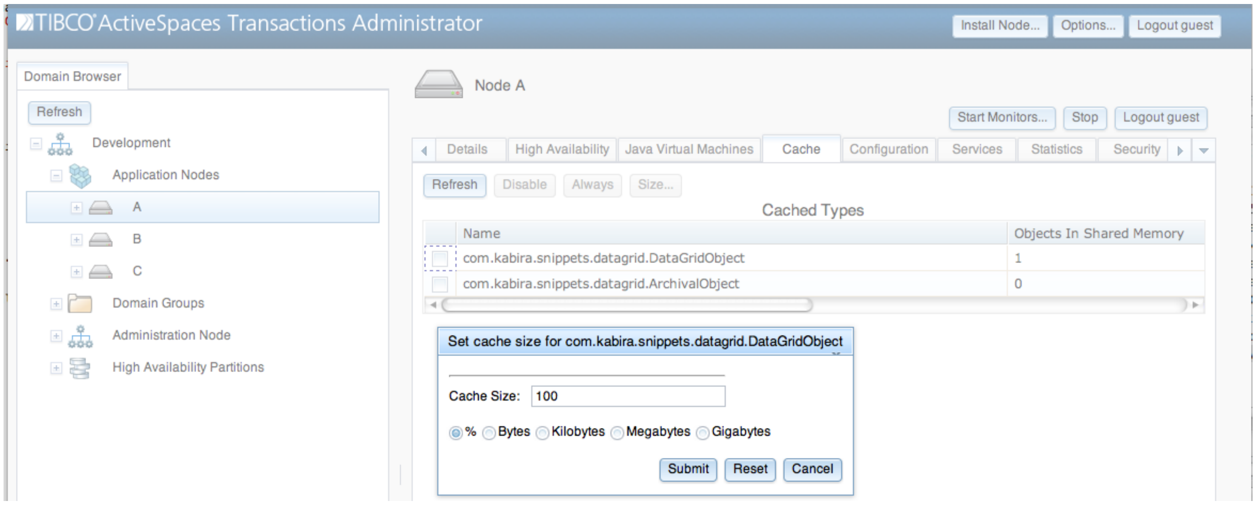
```
        metadata: 1336, 1424, spaces: [allocation=1032] [system=24]
        key com.kabira.snippets.sizing.ObjectSize$MyObject::ByName: 136, 240
        optimal allocationSpaceBytes = 1032
        event queue: 0, 0
        Total: 1472 1664
```

Transaction allocation pages are created during in-flight transactions to manage write and read images (snapshots) of the object. The memory used for in-flight transactions pages is equivalent to the size of the object fields. The transaction page is released when the transaction commits or aborts. The total amount of memory consumed for transaction pages should be multiplied by the number of concurrent transactions to get the total impact on shared memory size. For example, if a system is running at 1000 transactions / second, and each transaction creates a transaction page of a 50 byte object, the in-flight transaction log size is 1000 * 50, or 50,000 bytes.

Finally, any replica objects on a node consume shared memory. The amount of shared memory consumed by a replica object is the same as the shared memory consumed by the object on its active node, which can be calculated using the `com.kabira.platform.Type.memoryUsage()` method as discussed above.

Caching Distributed managed objects support caching a sub-set of the objects in shared memory. Cached objects consume the same amount of shared memory as non-cached managed objects. When cached objects are flushed their index data is also removed from shared memory.

The total shared memory consumed by cached objects can be explicitly controlled. The allocated cache size includes both object and index data in shared memory. The size of the shared memory cache can be specified as an absolute value or as a percentage of the shared memory available to the node. The cache size is set per node using TIBCO ActiveSpaces® Transactions Administrator. Figure 2.2 shows setting the cache size for the `com.kabira.snippets.datgrid.DataGridObject` class to `15%` of the total shared memory allocated on the A node.



Cached Types

	Name	Objects In Shared Memory
☐	com.kabira.snippets.datagrid.DataGridObject	1
☐	com.kabira.snippets.datagrid.ArchivalObject	0

Set cache size for com.kabira.snippets.datagrid.DataGridObject

Cache Size: 100

● % ○ Bytes ○ Kilobytes ○ Megabytes ○ Gigabytes

Figure 2.2. Setting cache size

Heap memory

JVM heap memory usage in TIBCO ActiveSpaces® Transactions follows normal JVM heap usage, with the following differences:

- Array fields in Managed objects only consume the size of an object reference (8 bytes).

- Managed objects have an additional, internal 24 byte field used as a shared memory identifier.
- POJO fields, for POJOs with the Transactional annotation, when transactionally locked and modified, will temporarily consume heap memory to log their before state. The memory used will be equivalent to the size of the fields before they are modified and is released when the transaction commits or aborts. The number of concurrent transactions should also be taken into account.

Process memory

A TIBCO ActiveSpaces® Transactions node consists of a small number of processes communicating through shared memory to provide the TIBCO ActiveSpaces® Transactions runtime services. The total size of the code that may be executed by these processes is approximately 100 megabytes. This is a system wide (per-server) cost, and not a per-process cost, because the code is contained in shared objects (.so) files.

Swap space

A typical Unix installation requires adding at least as much swap space as there is physical memory. However it is highly recommended that a machine running a TIBCO ActiveSpaces® Transactions system have enough physical memory so that it never needs to swap. Swapping runs at disk speeds, TIBCO ActiveSpaces® Transactions is designed to run from memory, at memory speeds.

Processing Units

- Clock speed

  Differences in processor speed have a direct linear effect upon the performance of application code. Faster processors will result in faster application execution.

- Multi-processor and multi-core

  Both the TIBCO ActiveSpaces® Transactions runtime and the JVM are designed to take advantage of multi-threading capabilities in the underlying operating system. A TIBCO ActiveSpaces® Transactions application designed for parallelism will take advantage of multiple processing units, increasing overall throughput.

Disk

- **Size**

  The TIBCO ActiveSpaces® Transactions product installation will make use of approximately 1 gigabyte of disk space.

  Deployed, each TIBCO ActiveSpaces® Transactions node's disk space is determined mostly by the size of the shared memory. By default this size is 512 megabytes and the shared memory is an ordinary file system file, which is memory-mapped by the TIBCO ActiveSpaces® Transactions runtime. There is also an option to use System V Shared Memory instead of a file. In this case, the shared memory does not use any disk space.

  Deploying the shared memory file in a networked file system (e.g. NFS) is not supported.

After system startup, TIBCO ActiveSpaces® Transactions , by default, will generate very little disk I/O, most of it involved in logging the invocation of administrative commands.

Application specific logging or generation of disk data also needs to taken into account when choosing disk size.

- **Number**

  By default, a single disk is capable of supporting a TIBCO ActiveSpaces® Transactions node.

  Disk I/IO speeds need to be considered when either change logging or application specific disk I/O will occur. If a single disk does not have the sufficient space or performance characteristics, the I/O may be spread across multiple disks either through configuration of the file locations, or by using a volume manager to present multiple disks as a single logical disk to TIBCO ActiveSpaces® Transactions .

- **Partitioning**

  TIBCO ActiveSpaces® Transactions does not have specific partitioning requirements.

  Using multiple partitions does not improve the performance characteristics of a single disk.

Network

Network speed affects both the throughput and latency when using highly available or distributed objects.

When a highly available object is modified, all of the object's data is sent to the remote node(s) when the transaction commits. The TIBCO ActiveSpaces® Transactions runtime attempts to minimize the number of separate packets by aggregating the data for multiple highly available objects that have been modified in a single transaction.

Distributed objects, depending upon configuration, and where they are being accessed may generate synchronous network I/O for each access.

For highly available and distributed objects, additionally network I/O is done between all involved nodes as part of transaction commit and abort processing. The size of the I/O is typically small, but it is a separate packet at commit/abort time.

A distributed application can saturate a Fast Ethernet (100 Mbits/second) network. It is recommended that Gigabit Ethernet be used.

Highly available and distributed objects, and the underlying TIBCO ActiveSpaces® Transactions support are designed to be used on a local area network (LAN) with low latency and high throughput. They also work are optimized to work over a wide area network (WAN) to support geographic redundancy.

High Availability

- **Number of machines**

Highly available objects exist in partitions that are shared between multiple nodes. It is recommended that these nodes be located on different machines. Multiple partitions may be hosted on a node, and nodes may act as both the active and replica roles for each other in an active/active configuration.

The number of partitions and the number of machines are chosen for both administrative and performance reasons.

- **Network interfaces**

  By default, TIBCO ActiveSpaces® Transactions uses a single network interface per node, but it may be configured to use multiple network interfaces.

3

Sizing Template

This chapter describes a template that can be used to estimate the system configuration for TIBCO ActiveSpaces® Transactions applications.

Memory Size

Add up the following sizes to determine the minimum physical memory size required.

- Shared memory size
- JVM heap memory size
- TIBCO ActiveSpaces® Transactions runtime executable size
- Operating system size
- Non-TIBCO ActiveSpaces® Transactions process sizes

Example:

A TIBCO ActiveSpaces® Transactions application using highly available objects with a single active node and a single replica node on separate machines.

```
Size of application highly available object - 4 kilobytes
Number of highly available objects persisted - 1000000
4096 * 1000000 = 4096000000 bytes  (approximately 4 gigabytes).

Number of transactional objects modified per transaction - 4
Amount of memory modified per object - 8192 bytes
Number of concurrent transactions - 64
4 * 8192 * 64 = 2097152 bytes

Primary Machine:
  70 Megabytes (runtime shared memory footprint)
4096 Megabytes (highly available objects persisted in shared memory)
   2 Megabytes (in flight transactions)
```

```
 100 Megabytes (TIBCO ActiveSpaces® Transactions
 process memory)
===========================
4268 Megabytes  + JVM memory + Operating system memory + Other processes memory

The backup machine has the same memory requirements as the primary machine.
```

Path length

Processing time for a unit of application work (e.g. request/response processing), less any time spent blocked waiting (e.g. Disk I/O, or waiting for a response from an intermediate system).

Latency

The elapsed time for servicing a unit of application work. This is the sum of the path length cost and any time spent blocked.

If the latency is larger than the required target then it can be reduced by one or more of the following items:

- Decrease the amount of path length work (simplify or optimize the application).
- Caching data in memory instead of retrieving it from an external resource (e.g. Disk or network).
- Increasing the performance of resources which are waited for (e.g. Faster disk or network).
- Increasing processor clock rate.

Message Throughput

In an ideal system, with a parallelizable load, the message throughput is simply the number of processors divided by the path length.

Network Utilization

When using High-availability, Managed object data is transparently copied across the network between nodes. The size of the object data copied is approximately the same size as the optimal allocationSpaceBytes reported by the memoryUsage() report shown earlier in the Shared Memory section of the Sizing Metrics chapter. There are also PDU headers sent.

- Header - 28 bytes
- SendMarshal - 112 bytes
- ResponseMarshal - 56 bytes
- TransactionInfo - 40 bytes

The following calculations may be used for estimating network traffic between each of the nodes that are part of the Managed object's High-availability partition:

For each Managed object create there is a request:

```
Header + SendMarshal + ObjectDataSize
```

and a response:

```
Header + ResponseMarshal + ObjectDataSize + TransactionInfo
```

Example for the create of Managed object of 1024 bytes:

```
 1024 + 28 + 112 + 1024 + 28 + 56 + 40 = 2312 bytes
```

For Managed object updates and Managed object multiple objects are sent in the same PDU. The request contains:

```
   Header + SendMarshal + (N * ObjectDataSize)
```

where N is the number of objects modified in the current transaction. This is limited by maximumPDUSizeBytes, which defaults to 1000000. Larger transactions will be spread across multiple network requests.

The response contains:

```
   Header + ResponseMarshal + TransactionInfo
```

Example for the update of Managed object 1 (1024 bytes) and Managed object 2 (2048 bytes) in the same transaction:

```
       28 + 112 + 1024 + 2048 + 28 + 56 + 40 = 3336 bytes
```

Disk Size

- The space required by the TIBCO ActiveSpaces® Transactions installation (typically less than 2 Gigabytes).
- If using file based shared memory, the space required by the shared memory file. This size is configurable and set at node installation time. System V shared memory does not use disk space for storing the shared memory.
- Swap space on the system, at least the same size as physical memory.
- Space for node log files. In typically production operation not much space is needed, but these files can grow over time, and can grow at faster rates if extra system tracing is enabled, or if there are many exceptions or deadlocks in the system. 1 Gigabyte should be a sufficient amount of space to start with.
- Space for application logging. This is application dependent.

Index