# TIBCO Enterprise Runtime for R

# Performance Guide

http://www.tibco.com

**Global Headquarters**
3303 Hillview Avenue
Palo Alto, CA 94304
Tel: +1 650-846-1000
Toll Free: 1 800-420-8450
Fax: +1 650-846-1005

## Contents

## Executive Summary

TIBCO Enterprise Runtime for R (TERR) was architected from the ground up as a high-performance, enterprise-quality runtime for the R language. One of the key advantages of TERR vs. the open source (OS) R engine is a significantly faster average performance on larger data sets.

Prospective users of TERR often ask, “How much faster is TERR than Open Source R?” Because R is a comprehensive programming language with a large number of built-in functions and contributed packages, it is impossible to give a single, simple answer to that question. Instead, we recommend that potential users of TERR evaluate TERR’s performance on their own R language analytics to evaluate its performance and other advantages for themselves. The TERR Developer’s Edition is available as a free download to enable this, and TIBCO is happy to assist with these efforts.

However, prospective users of TERR often request a starting point for their own performance tests and some sample results as guidance. This Guide provides sample performance results for TERR vs. Open Source R.

Overall, when compared to OS R:

- For many common operations on small to moderate size data sets, TERR is 2-10x as fast as OS R
- When applied to larger data sets, for common operations such as model scoring, or on complex, real-world scripts where TERR’s more efficient memory management becomes critical, TERR can be tens to a hundred or more times as fast as OS R.

## Introduction

TERR, a key component of Spotfire Predictive Analytics, is an enterprise-grade analytic engine that TIBCO has built from the ground up to be fully compatible with the R language, leveraging our long-time expertise in the closely-related S+ analytic engine. This compatibility allows customers to continue to develop in open source R, but then to integrate and deploy their R code on a commercially-supported and robust platform without the need to rewrite their code.

Prototypes are often developed in R, but then typically re-implemented in another language for production purposes because the open-source R engine was not built for enterprise usage. TERR brings enterprise-class scalability and stability to the agile R-language. Statisticians can share their analyses through TIBCO Spotfire Statistics Services or through the TERR engine directly embedded into another application.

TERR enables customers to iterate rapidly from prototyping to production without wasting time and effort recoding and retesting their analyses, allowing them to respond more rapidly to opportunities and threats, and to integrate standardized predictive analytics easily and consistently across an organization

TERR's architectural advantages include:

- The modern architecture was designed by computer scientists with many years of experience working with the S/R language and statistical engines.
- The TERR interpreter was written entirely in C++ to maximize performance.
- Memory management was designed to ensure robust handling of large data, eliminating unnecessary copying of the data, and ensuring that TERR performs reliably and predictably with increasing data sizes. In contrast, OS R's memory management doesn't reclaim memory as well as TERR, so memory use can grow faster, leading to out-of-memory crashes, as well as very non-linear performance (due to more garbage collections, and increased swapping).
- Extensive work has been done to implement thorough and robust error handling for enterprise deployments.

## Approach

The general approach was to aggregate publicly-available benchmarks with simple, internally developed benchmark scripts, based on our general experience of customer usage.

- All performance tests compare a release of TERR with the corresponding release of the open source R engine.
- The open source R engine and any necessary packages were downloaded from CRAN (http://cran.us.r-project.org/) and not modified in any way.
- While the performance tests were run on various hardware combinations, each individual test to compare TERR vs. OS R was run on the same hardware, to ensure meaningful comparisons.

## Results

### Test 1: Row-by-row processing in a for loop

While R programmers are encouraged to take advantage of R's vectorization, a new R programmer will often use a simple loop as a structure familiar from other programming languages. In this test, a `for` loop iterated through a long data frame, and did a trivial operation on each record:

```
for (i in seq(1,length=nrow(df))) {
…process each customer record…
}
```

**Results: TERR was ~35x faster for 50K rows, and 150x faster for 500K rows**, illustrating both the performance gains TERR can provide on real-world, unmodified, unoptimized R scripts, and the advantages of its superior memory management.

More information on this test is included in Michael Sannella's presentation for useR 2013.

### Test 2: Fitting and Scoring Generalized Linear Models (GLM)

In this test, GLM models were fitted and scored on randomly-generated data using TERR and OS R. (See the Appendix for the R scripts used.)

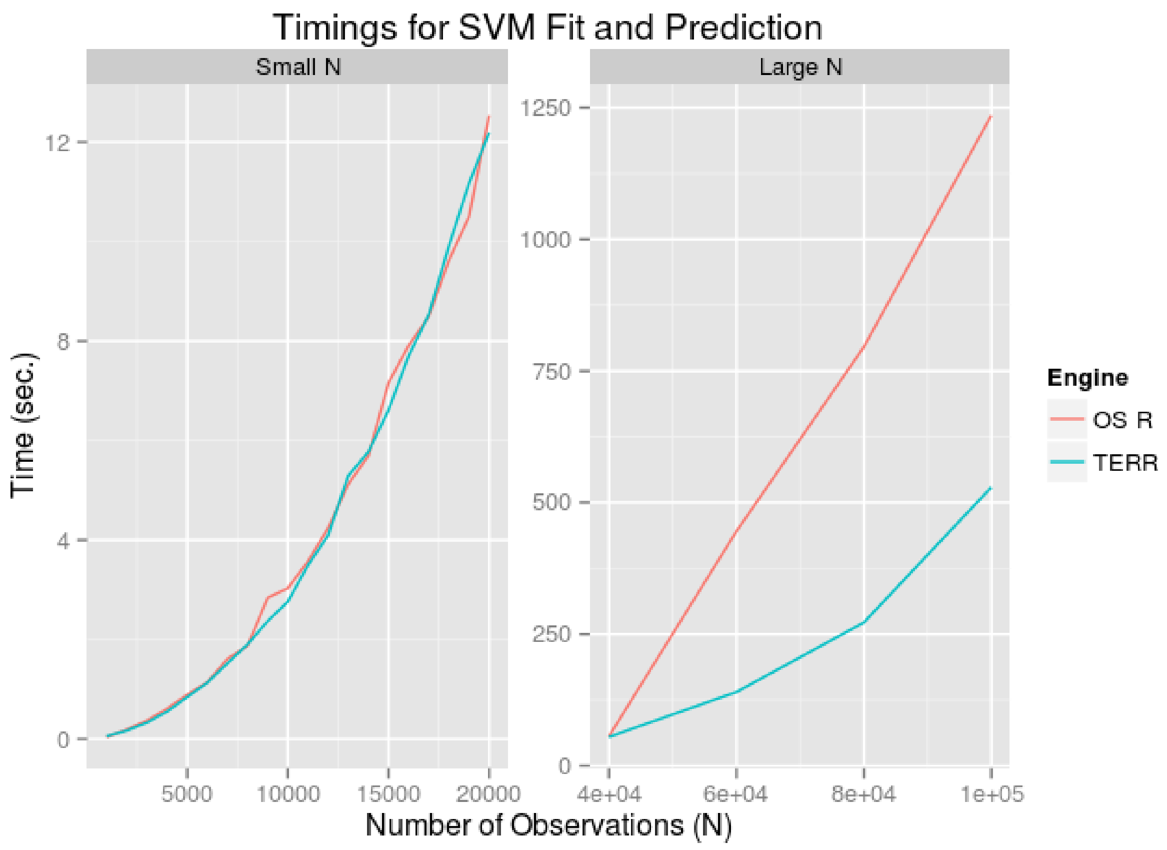
**Results (Elapsed Time)**

| | OS R | TERR | Speedup |
|---|---|---|---|
| **Model Fitting on 5 M rows** | 107.1 sec | 17.5 sec | **6.1 x** |
| **Model Scoring on 20M rows** | 84.2 sec | 1 sec | **84.2 x** |

## Test 3: Predictions using SVMs from the e1071 package

This analysis compares the performance of TERR vs. OS R for generating test data, and then applying `svm/predict` from the e1071 package. The results are plotted in the visualizations below, with elapsed time in plotted on the y axis, and the number of observations TERR plotted on the x axis. (See the Appendix for the R script used.)

For smaller numbers of training samples (less than 20K), the performance is comparable (see the left graph below).



For larger numbers of samples (show in the right graph above), TERR becomes faster than OS R, and its performance advantage increases as the number of iterations increase—again illustrating the benefits of TERR's stronger memory management.

## Test 4: Public R Benchmarks from AT&T Research

A number of public benchmarks have been provided by Simon Urbanek at AT&T Research (and member of the R Core team): http://r.research.att.com/benchmarks/

While these benchmarks mainly test matrix operations, they are frequently cited, and so they are included here. TERR uses the Intel MKL libraries, while OS R uses non-optimized LAPACK libraries.

| Name | Description | TERR 2.5.0(s) | R 3.0.2(s) | Speedup |
|---|---|---|---|---|
| **Transpose Matrix** | Transpose a 2500 by 2500 matrix. | 0.18 | 1.08 | 6.00 |
| **Matrix Power** | Exponentiate a 2500 by 2500 matrix | 0.39 | 1.92 | 4.92 |
| **QuickSort** | Quicksort on a vector with length 7,000,000 | 0.42 | 1.89 | 4.51 |
| **Cross Product** | Cross product of a 2500 by 2500 matrix | 0.54 | 4.99 | 9.25 |
| **Matrix Solver** | Solve a 2000 by 2000 matrix | 0.53 | 4.39 | 8.28 |
| **FFT** | FFT on a vector of length 2,400,000 | 0.1 | 0.84 | 8.44 |
| **Eigen** | Calculate the eigenvalues and eigenvectors of a 600 by 600 matrix | 0.5 | 1.17 | 2.35 |
| **Determinant** | Take the determinant of a 2500 by 2500 matrix. | 0.42 | 3.89 | 9.27 |
| **CholeskyR25** | Decompose a 3000 by 3000 matrix | 1.16 | 10.87 | 9.37 |
| **Matrix Inversion** | Inverting a 1600 by 1600 matrix | 0.39 | 3.38 | 8.66 |
| **Fibonacci** | Calculating the first 3,500,000 Fibonacci numbers | 0.7 | 1.32 | 1.88 |
| **Hilbert Matrix** | Create a 3000 by 3000 Hilbert Matrix | 0.24 | 0.40 | 1.65 |
| **GCD Recusion** | Calculate the GCD of two numbers recursively | 0.17 | 0.87 | 5.14 |
| **Toeplitz Matrix** | Create a 1000 by 1000 Hilbert Matrix | 1.73 | 3.66 | 2.12 |
| **Escoufier** | Use Escoufier’s Method to calculate equivalent vectors of length 45 | 0.51 | 0.59 | 1.15 |

On average, **TERR is 5.5x faster** than OS R for these operations in these tests.

## Test 5: Public R Benchmarks from Revolution Analytics

Revolution Analytics provides a set of benchmarks on their website (see link in References section). The results of these benchmarks on TERR and OS R are shown below.

| Name | Description | TERR 2.5.0 (s) | R 3.0.2 (s) | Speedup |
|---|---|---|---|---|
| **Matrix Multiply** | Multiply a 10,000 by 5,000 matrix with a 5,000 by 10,000 matrix | 6.82 | 67.16 | 9.9 |
| **Cholesky** | Using Cholesky Factorization on a 10,000 by 5,000 matrix | 8.2 | 81.90 | 10.0 |
| **SVD** | Perform SVD on a 10,000 by 2,000 matrix. | 5.55 | 35.44 | 6.4 |
| **PCA** | Perform PCA on a 10,000 by 2,000 matrix | 11.16 | 113.46 | 10.2 |
| **LDA** | Performing Linear Discrimate Analysis on a 10,000 by 2,000 matrix. | 13.83 | 111.02 | 8.0 |

On average, **TERR is 8.8x faster** than OS R for these operations in these tests.

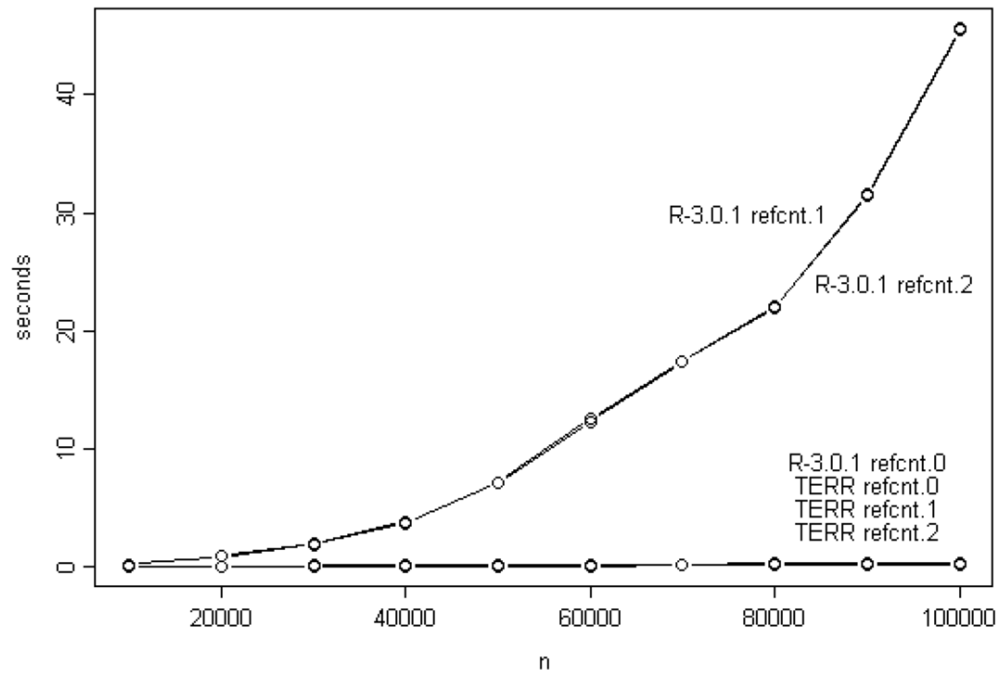
## Test 6: Reference Counts

Both OS R and TERR use reference counts to efficiently track how many times an object has been referenced. If an object has only one reference, then it can be safely modified by a particular bit of code without copying it. Unnecessary copying of objects consumes system resources, and it can slow processing down and eventually lead to crashes.

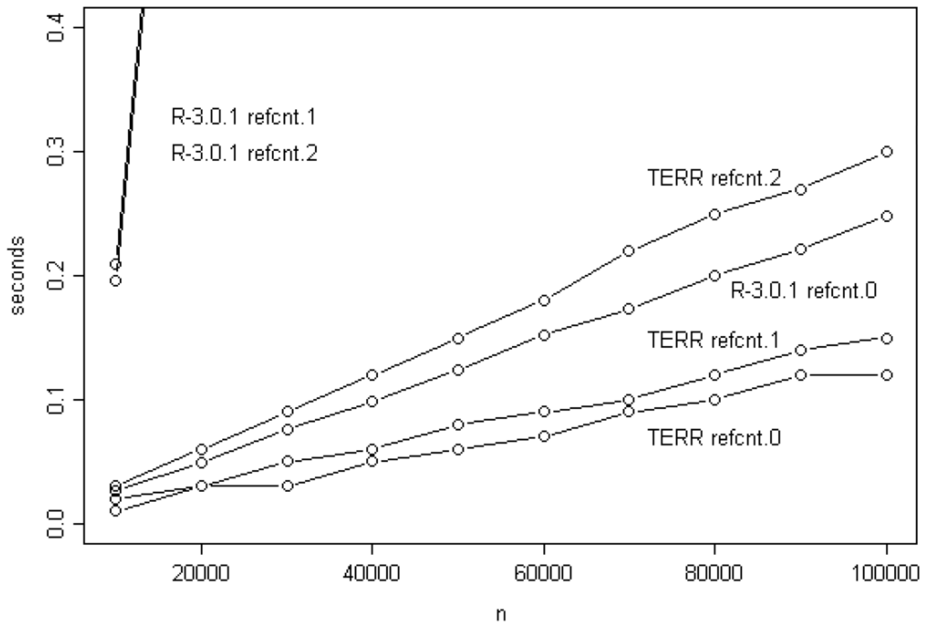
As Michael Sannella, the senior architect for TERR presented at useR 2013, TERR can very efficiently track large numbers of reference counts to particular data objects, eliminating unnecessary copying. To illustrate this, 3 tests were devised (the code for these tests are available at the link above):

- refcnt.0: as the control for comparison, this script does not add a reference count.
- refcnt.1: this script temporarily adds a second reference to an object within a `for` loop.
- refcnt.2: this script temporarily adds a second reference to an object via a function call.

These tests illustrate the performance benefits of TERR's efficient memory management on larger data sets.



**Results of reference count tests**



**Results of reference count tests (same as previous graph, but zoomed on the y axis)**

## References

- Overview of TERR on the TIBCO Spotfire website: http://spotfire.tibco.com/terr
- TERR Community Site: https://community.tibco.com/wiki/tibcor-enterprise-runtime-r
- Download the TERR Developer's Edition: http://tap.tibco.com
- Memory Management in TERR (presented at useR 2013): http://www.slideshare.net/loubajukyorgan/sannella-use-r2013terrmemorymanagement
- AT&T Research Benchmarks for R: http://r.research.att.com/benchmarks/
- Revolution Analytics Benchmarks for R: http://www.revolutionanalytics.com/revolution-revor-enterprise-benchmark-details

## Appendix

### Code for Test 2: Fitting and Scoring Generalized Linear Models (GLM)

Model Fitting:

```
n <- 5000000
x1 <- rnorm(n)
x2 <- rbinom(n,1,.5)
x3 <- rbinom(n,1,.5)
b0 <- 1; b1 <- 1.5, b2 <- 2; b3 <- 2.5
y <- binom(n,1,invlogit(b0 + b1*x1 + b2*x2 + b3*x3))
system.time(fit <- glm(y ~ x1 + x2 +x3, family=binomial(link=”logit”))

Model Scoring:
n <- 20000000
x1 <- rnorm(n)
x2 <- rbinom(n,1,.5)
x3 <- rbinom(n,1,.5)
system.time(prediction <- predict(fit,newdata = data.frame(x1,x2, x3))
```

### Code for Test 3: Predictions using SVMs from the e1071 package

```
library(e1071)

p <- 2
sigma <- 1
meanpos <- 0
meanneg <- 3
res=c()
for(n in c(seq(from=1000, to=20000, by=1000), 40000, 60000, 80000, 100000)){
t0 <- proc.time()[3]
npos <- round(n/2)
nneg <- n-npos
xpos <- matrix(rnorm(npos*p,mean=meanpos,sd=sigma),npos,p)
xneg <- matrix(rnorm(nneg*p,mean=meanneg,sd=sigma),npos,p)
x <- rbind(xpos,xneg)
y <- matrix(c(rep(1,npos),rep(-1,nneg)))

ntrain <- round(n*0.8) # number of training examples
tindex <- sample(n,ntrain) # indices of training samples
xtrain <- x[tindex,]
xtest <- x[-tindex,]
ytrain <- y[tindex]

ytest <- y[-tindex]
istrain=rep(0,n)
istrain[tindex]=1

model=svm(xtrain,ytrain)
ypred=predict(model, xtest)
t1 <- proc.time()[3]
time.pro <- t1-t0
res=c(res,time.pro)
}
```