

SORRENTO MANUAL

**by
Aziz Gulbeden**

September 13, 2004

Table of Contents

SORRENTO – SELF-ORGANIZING CLUSTER FOR PARALLEL DATA-INTENSIVE APPLICATIONS	1
OVERVIEW	1
COPYRIGHT	1
SORRENTO WALKTHROUGH - INSTALLATION	3
1. REQUIRED LIBRARIES	3
2. SOURCE CODE	3
3. BUILDING SORRENTO LIBRARIES	3
4. BUILDING SORRENTO DIRECTORY SERVER	3
5. BUILDING STORAGE PROVIDER AND OTHER UTILITIES	3
SORRENTO WALKTHROUGH – DEPLOYING SORRENTO.....	5
1. STARTING THE NAMESPACE SERVER	5
2. STARTING STORAGE PROVIDER(S)	5
3. MONITORING THE SORRENTO DEPLOYMENT.....	5
4. TESTING THE SORRENTO DEPLOYMENT	5
SORRENTO WALKTHROUGH – USING SORRENTO IN APPLICATIONS THROUGH SORRENTO API	6
SORRENTO WALKTHROUGH – USING SORRENTO IN APPLICATIONS THROUGH KERNEL SWITCH	8
STRUCTURE OF THE SORRENTO CODE.....	9
1. SERVER DIRECTORY:	9
1.1 <i>Classes Used By The Client Side Applications:</i>	9
1.2 <i>Classes Used Internally By The System</i>	10

SORRENTO – Self-Organizing Cluster for Parallel Data-Intensive Applications

Overview

The goal of the Sorrento project is to build a self-organizing storage system upon the cluster architecture, with emphases on four aspects: programmability, manageability, performance, and availability.

Clusters provide a cost-effective computing platform, and incremental scalability. Sorrento is built upon the cluster architecture and aims to provide more efficient usage of storage devices and I/O bandwidth.

Sorrento is designed for cluster applications that need to access large amounts of data and whose working set does not fit into the memory. Internet services, data mining, stream-media services are among the applications that can benefit from such a system.

Sorrento project is headed by Hong Tang and Professor Tao Yang. The code has mainly been written by Aziz Gulbeden, William D. Strathearn, Hong Tang, Jingyu Zhou with much help from other students including Lingkun Chu, Zhongnan Shen.

The code is written in C/C++, and targeted for Linux based clusters.

Sorrento project is partially supported by NFS and Ask Jeeves Inc.

Copyright

Copyright (c) 2001, 2002 The Regents of the University of California
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by the University of California, Santa Barbara.
4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND THE AUTHORS ``AS IS''
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS

BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY,
OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Sorrento Walkthrough - Installation

This document describes the installation steps of Sorrento for a Linux system.

1. Required Libraries

The “gsl” and “db3” libraries need to be present in the system for Sorrento to compile and run.

You can use the command “rpm -qi gsl”, “rpm -qi db3” to check if these libraries are installed. If you do not have root access, you may install them in your home directory and link Sorrento to them during compilation by modifying the file “cbs/src/config.lib” accordingly.

2. Source code

The source code can be downloaded from: <http://www.cs.ucsb.edu/projects/sorrento/> .

3. Building Sorrento Libraries

1. Extract the archive in the directory where you want to install Sorrento in.
2. cd into “cbs/src/shared”
3. Execute “make init” which will create the necessary directories for Sorrento installation. (in “cbs/src/shared”)
4. Build the libraries used by Sorrento through the command “make lib” (in “cbs/src/shared”)

4. Building Sorrento Directory Server

1. cd into “cbs/src/daemon”. (You can use the command: “cd ../daemon/”)
2. Issue the command “make lib” (in “cbs/src/daemon”).
3. cd into “cbs/src/daemon/dir_srv”. (You can use the command: “cd dir_srv/”)
4. Create the executable through “make” command.

5. Building Storage Provider and Other Utilities

1. cd into into “cbs/src/daemon/server”. (You can use the command: “cd ../server”)
2. Create the executables through “make” command.
3. cd into into “cbs/src/daemon/tools”.

4. Create the executables through “make” command.

Now all the executables should be ready for deploying Sorrento. Read the next section for executing Sorrento.

Sorrento Walkthrough – Deploying Sorrento

This document explains how to run Sorrento after the executables are built.

1. Starting the Namespace Server

1. cd into “cbs/src/daemon/dir_srv”.
2. Start the directory server through the command “./dir_srv -f dirsrv.conf”.

2. Starting Storage provider(s)

1. cd into “cbs/src/daemon/server”.
2. Edit the file “sorrento.conf”. Here the Ident and Channel should be the same as the ones in the configuration file used by the directory server (i.e. “./dir_srv/dirsrv.conf”). Modify Home, Partitions¹ and other parameters in the file if necessary. Save and exit.
3. Prepare the storage provider directories through the executable sor_install located at “cbs/src/daemon/tools” through the command:
“./tools/sor_install -fig sorrento.conf”
(execute this from the directory “cbs/src/daemon/server”).
4. Start the storage provider through the command “./main sorrento.conf”.

3. Monitoring the Sorrento Deployment

1. cd into “cbs/src/daemon/server”.
2. Start the Sorrento monitor utility through the command “./t4 sorrento.conf”.
This program will display the resources in the current deployment, and their loads.

4. Testing the Sorrento Deployment

1. cd into “cbs/src/daemon/server”.
2. Start the Sorrento shell through the command “./sor_shell -f sorrento.conf”.
This program will open a shell that allows basic file operations to be performed and tested.

¹ Using more than one partition in the current version might cause problems.

Sorrento Walkthrough – Using Sorrento in Applications Through Sorrento API

This document explains how Sorrento can be used by linking applications directly to Sorrento. Here is a sample C++ program that makes file system calls through Sorrento:

```
# include <stdio.h>
# include <signal.h>

# include "sor_client.h"
# include "sor_fs.h"
# include "sor_file.h"
# include "subscriber.h"

using namespace std;

static Subscriber *sub = NULL;
static SorClient *scli = NULL;

static void output_error(const char *cmd){
    printf("%s failed: %s.\n", cmd, scli->str_error().c_str());
}

int main(int argc, char *argv[]){
    const int blockSize = 4;
    char buf[blockSize+1] = "ABCD";
    char objName[16] = "minimal.test";
    if (argc < 3){
        printf("Usage %s <Conf-File> <DeploymentID>\n", argv[0]);
        printf("e.g. %s sorrento.conf Mayo\n", argv[0]);
        return -1;
    }
    char *config=argv[1];
    char *ident = argv[2];
    SorFile *psf;
    off_seg_t written_len;

    signal(SIGPIPE, SIG_IGN);
    SorrentoConf sconf(config);
    sub = new Subscriber(sconf);
    sub->start();
    printf("Starting the proxy, please wait ...\n");
    sleep(3); // wait for 3 seconds.
    SorFS sorfs(sub, string(ident));
    scli = new SorClient(&sorfs);

    psf=scli->create(objName);

    if (psf == NULL){
        output_error("Create");
        return -1;
    }

    if (scli->write(psf, 0, buf, blockSize, written_len)==false)
        output_error("Write");
}
```



```
if (scli->close(psf) == false)
    output_error("Close");

sub->stop();
sub->finalize();
delete sub;
delete scli;
return 0;
}
```

This program is present in the codebase in the file:

“cbs/src/expr/remexec/minimal.cc”.

Use the command “make minimal” to build it and “./minimal sorrento.conf Mayo” to execute it. The directory server and at least one storage provider should be running in order to perform the operations successfully. The program creates a file named “minimal.test” in Sorrento and writes the string “ABCD” into the file.

Sorrento Walkthrough – Using Sorrento in Applications Through Kernel Switch

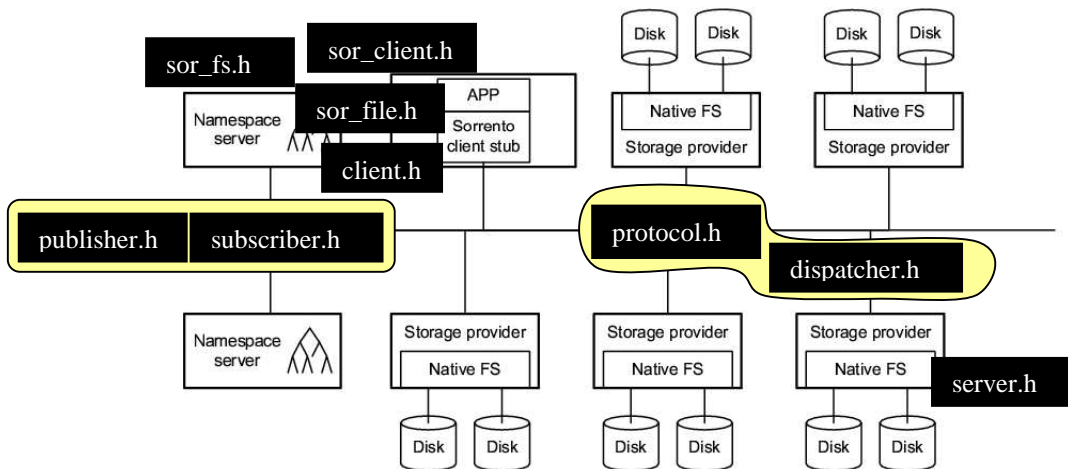
This document explains how Sorrento can be used by linking applications directly to Sorrento.

(to be written by Jingyu or Bill)

Structure of the Sorrento Code

This document provides an overview of the components in the Sorrento system by explaining their functionalities and how different modules interact with each other.

1. Server Directory:



1.1 Classes Used By The Client Side Applications:

SORRENTO CLIENT

Header file:

sor_client.h

Description:

Applications use this header file to access Sorrento. sor_client provides basic file system functionality, it allows making operations on directories and files.

Example:

See “cbs/src/expr/remexec/micro_ben2.cc” for an example.

SORRENTO FILE SYSTEM

Header File:

sor_fs.h

Description:

Maintains the information regarding the file system.

SORRENTO FILE

Header file:

`sor_file.h`

Description:

This module represents the way a file is stored in the system. Small files that can be sent within a UDP packet are stored together with their metadata header. For larger files metadata and the segments are separated. The metadata block contains the GUIDs of the data segments.

SUBSCRIBER

Header file:

`subscriber.h`:

Description:

The messages published by the nodes in the system are received by the `subscriber` module of the other components. This module keeps the membership information.

1.2 Classes Used Internally By The System

client.h: This module is used by the `sor_file` module, it allows `sor_client` to access the system at the granularity of segments (not files). This appears as a separate module than `sor_client` because it hides the details of the system from applications. Also any other module might use `client.h` if it needs to read data from, or write data to another storage server (as in the case of replication or migration).

protocol.h: Protocol allows the messages to be sent between two hosts in the system. The client's requests (such as open, read, remove file) are sent to the storage servers through this module. Similarly, servers use this module to interact with each other for purposes such as notifying homelost about a change in a locally stored segment or for replication requests and notifications.

dispatcher.h: Storage servers listen to the network using this module, it listens on both a TCP port and a UDP port and forwards the incoming requests to the corresponding module in the storage server part.

publisher.h: Periodically the storage providers announce their presence and their load to the other components through a multicast channel.

host load.h: This keeps the information that is being published.

server.h: The `server` module provides storage to the `client`. It communicates to the `client` using `protocol` module through `dispatcher`, and membership information is kept using the `subscriber` module. The `server` module contacts other servers to push or pull data from them in order to locate data, or to replicate it.

srcatalog.h: This module stores information about the segments whose homehost is the current host but it is stored at another storage provider.

srcatalogupdater.h: Catalog updater periodically pulls data from other providers asking for objects that map to the current host through consistent hashing.

host_map.h: Hostmap is used for translating object ids to host ids. If the homehost of a particular segments is being searched, the `hostmap` provides the address of the host that the segment maps to.

sor_fs.h: Handles the communication with the metadata server.

sorrento_conf.h: Reads the configuration file that provides the setup information for a Sorrento session.

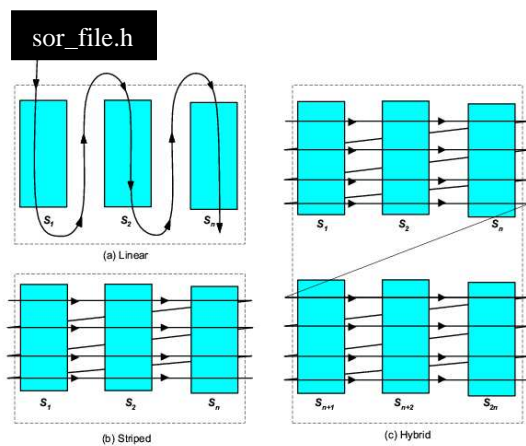
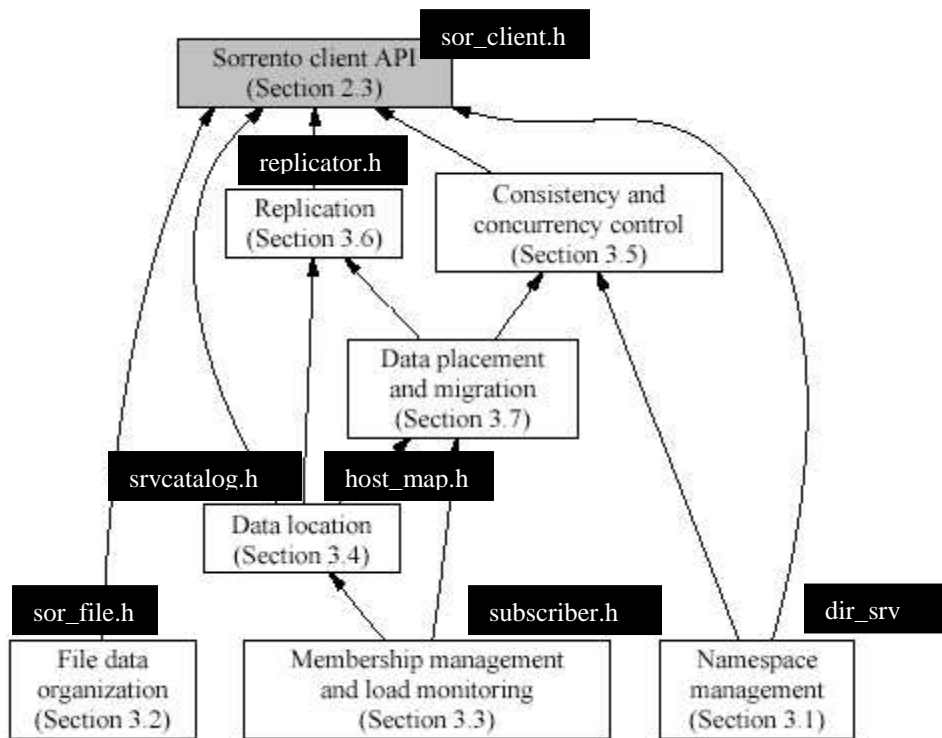
sor_dll.h: A doubly linked list.

sorrento_types.h: Contains the objects that are used commonly among different modules.

srv_callback.h: An interface to allow server to be notified about the host joins and leaves.

replicator.h: The replication module that keeps track of the replication degrees of the segments and where they are replicated at for the segments that map to the corresponding server. If the replication degree requirement is not satisfied, it chooses a new replica site, and if the versions of the replicas are not up to date, it notifies them to get the latest updates from the server that has them.

sor_shell.cc: A terminal to test functionality of Sorrento through commands.



Verfs Directory:

