CONTRACT NUMBER 508830

# DEISA

## DISTRIBUTED EUROPEAN INFRASTRUCTURE FOR SUPERCOMPUTING APPLICATIONS

**European Community Sixth Framework Programme**
RESEARCH INFRASTRUCTURES
Integrated Infrastructure Initiative

Proof of functionality of the general distributed file systems AFS and AVAKI.

Deliverable ID: D-SA2-1B

**Due date: October 30, 2004**
**Actual delivery date: November 24, 2004**
**Lead contractor for this deliverable: RZG, Germany**

**Project start date : May 1st, 2004**
**Duration: 5 years**

| **Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)** | | |
|---|---|---|
| **Dissemination Level** | | |
| **PU** | Public | |
| **PP** | Restricted to other programme participants (including the Commission Services) | X |
| **RE** | Restricted to a group specified by the consortium (including the Commission Services) | |
| **CO** | Confidential, only for members of the consortium (including the Commission Services) | |

# Table of Content

Project and Deliverable Information Sheet ..... 2
Document Control Sheet ..... 2
Document Status Sheet ..... 2
Document Keywords and Abstract ..... 2
Table of Content ..... 2
List of Figures ..... 3
1. Introduction ..... 4
1.1 Executive Summary ..... 4
1.2 References and Applicable Documents ..... 4
1.3 Document Amendment Procedure ..... 4
1.4 List of Acronyms and Abbreviations ..... 4
Description and Evaluation of the selected file systems ..... 5
2. ..... 5
2.1 Introduction ..... 5
2.2 AFS ..... 5
2.3 AVAKI ..... 6
3. Summary and recommendation ..... 7
4. Technical Annex : Evaluation of Avaki Data Grid v. 5.0 ..... 9

## List of Figures

Figure 1……..Accessibility of the AFS-cell deisa.org

# 1. Introduction

## 1.1 Executive Summary

The Service Activity 2 within the DEISA project deals with the connectivity of all DEISA-sites on the filesystem level. Two strategies are pursued in parallel:

- Deploying IBM's GPFS. See deliverable D-SA2-1A.
- Implementing a distributed filesystem structure for heterogeneous environment.

This document gives an overview of the two pre-selected filesystems AFS and AVAKI for DEISA. It reports about their stability and how they can be integrated into DEISA.

## 1.2 References and Applicable Documents

[1] http://www.deisa.org

[2] Deliverable D-SA2-1A

[3] Internal Avaki Evaluation Report. [appended as Technical Annex]

[4] http://www.nfsv4.org

[5] http://www.lustre.org

## 1.3 Document Amendment Procedure

Not applicable.

## 1.4 List of Acronyms and Abbreviations

| | |
|---|---|
| **AFS** | Andrew File System, used in the open-source implementation OpenAFS |
| **AIX** | Advanced Interactive eXecutive ((IBM's derivative of UNIX OS) |
| **AVAKI** | Avaki Data Grid, proprietary middleware to share files from Avaki |
| **FS** | File System |
| **GPFS** | General Parallel File System, proprietary FS from IBM |
| **HTTP** | Hyper Text Transport Protocol, the base protocol of the internet. |
| **HPS** | High Performance Switch |
| **LPAR** | Logical PARtition (subset of a larger system) |
| **NFS** | Network File System, a very common filesystem. |
| **NREN** | National Research and Education Network |
| **OS** | Operating System |
| **P655, P690** | High performance computing nodes built by IBM. |

# 2. Description and Evaluation of the selected file systems

## 2.1 Introduction

Common file systems provide one of the foundations of the integrated DEISA environment. The idea behind having a common file system is that deeper integration of the DEISA sites will make it easier for developers to write upper layer software spanning the whole DEISA infrastructure.
The two evaluated solutions AVAKI and AFS are not comparable as such, since AFS is a widely used and mature product, used in many scenarios, while on the other hand AVAKI is a new product aimed at a slightly different usage. It aims at data integration within companies, i.e. creating a single composite view of different data-sources such as files or databases.
The functionality which was expected from the two systems include easy installation, usage, reliability, transparency to the user.
AFS was installed and used in a test-environment between three of the four core-sites (CINECA, IDRIS, RZG) for evaluation of its feasibility.
AVAKI, on the other hand, was tested on its basic functionality and usability between two sites (IDRIS, RZG) only.

## 2.2 AFS

**Test systems of the three of the four core-sites**

IDRIS has three p655 nodes, two are equipped with 4 CPUs and one with 8 CPUs dedicated to the DEISA test-environment. These machines are connected internally with an HPS. Externally all three machines are connected to the DEISA-network with a 1Gigabit ethernet-interface.

RZG has a full p690 node divided into four LPARs having 8 CPUs each dedicated to the DEISA test-environment. The four machines are connected internally and externally with the DEISA –network via a 1 Gigabit ethernet-interface.

CINECA has two p690 LPARs with 8 CPU each dedicated to the DEISA test environment. The two machines are connected internally and externally with the DEISA –network via a Gb-interface.

**Network**

Currently the test environment is using the already installed 1 Gbit/s DEISA wide area network (see deliverable D-SA1-1.1).

**Software and Installation**

AFS is a global distributed file system which is organized in cells. Each cell is a distinct unit of administration with its own user and file space management, but all cells may be visible simultaneously in the AFS file-tree on each client. An own AFS cell named “deisa.org” has been created for DEISA, the data are visible under “/afs/deisa.org”.
AFS fileservers for this cell are presently running at RZG and CINECA, while the clients are installed in RZG, CINECA and IDRIS. AFS can be used as a failsafe way to provide

software for HPC-applications since in case of a breakdown of one fileserver, the client software automatically switches over to the next fileserver which contains a copy of the software. It can also be used for user home directories with the advantage that these home directories are not only visible inside the DEISA test environment, but also on any other AFS client. AFS client-software is available on a broad variety of platforms: most Unix systems, Windows and MacOS. On other systems it can be made visible by means of an AFS/NFS translator. Its design as a world wide file system guarantees security and connectivity in wide area environments as well as in local area networks.

**Usability**

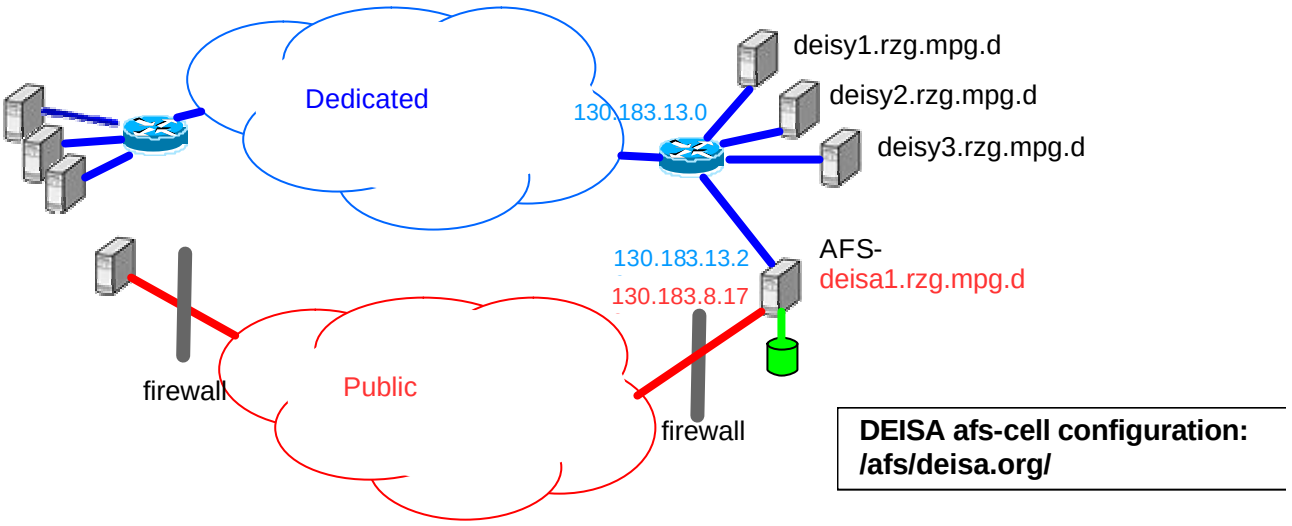



**Figure 1: Accessibility of the AFS-cell deisa.org**

The initial DEISA test environment lives in a nearly private network between the dedicated test machines of the core partners. The routing prevents access of data from outside this network, therefore other AFS cells than “deisa.org” are not visible on the dedicated test machines, but the data of this cell are visible outside the test environment because the AFS servers have a secondary network connection. In the 1st phase of the tests when the authentication of users and batch jobs by means of certificates is not yet in place the AFS credentials are automatically created based on the user’s unix userid. Therefore the user can access the data in the AFS cell “deisa.org” as if it would be a local file system. In the outside world, of course, he has to authenticate by his Kerberos password.

## *2.3 AVAKI*

Since the full internal AVAKI report is enclosed, only a short summary is given here.

**Test Systems of the core-sites IDRIS and RZG**

AVAKI itself was tested only at IDRIS and RZG, both locally and between the two sites.

IDRIS provided as IBM machines a p655 with 8 CPUs and a 8 CPU Power3-node, for Linux a single processor server.

RZG facilitated a 2 CPU LPar of a p690, 5 Linux servers, three with 2 CPUs and two with a single processor.

Details can be found in the enclosed internal report.

**Network**

Both sites have used a mix of 100 Mbit/s and 1 Gbit/s network links. For the interconnect between IDRIS and RZG, the public NRENs network were used.

**Software and Installation**

AVAKI itself needs only one server to run in a minimal configuration. However distributing the services over different servers gains performance benefits.
The administrative effort to connect different domains is significant and not straightforward. While installing the testbed many pitfalls were encountered.
For more details see the enclosed internal report.

**Usability**

From the user's point of view, a DEISA file-space implemented with AVAKI will be seen as an NFS mount or as web-space using an http-browser. Both methods can be used under UNIX to access the data.
The configuration of both the server and the client has to be carried out by employing a web interface and the command-line. A user, who wants to access the AVAKI filespace has to use both of them at the present state of AVAKI.
Using AVAKI as a NFSv3 exporter, one inherits all security issues of NFSv3, and the mount and logon procedure is cumbersome.
Beside exporting files, AVAKI is also able to map file-access to database-queries. This is done by special files which contain predefined SQL-statements. If the user accesses such a file the SQL-statement is passed on to the database and the result is written in another predefined, real file. This can simplify database access in some cases, but is usually too static for day to day use. Also the results are cached so that the user has to be wary not to get the result of a previous database-query.
Refer to the enclosed internal report for more details.

# 3. Summary and recommendation

In this document, two very different types of software have been evaluated and thus cannot be compared directly, but only in their ability to serve for the purposes of DEISA.

Summed up, it can be concluded that AFS seems fit for serving DEISA as a global file system on most of the installed computer architectures, as an alternative if high performance solutions are not available.
AVAKI, on the other hand is a new product and does not appear at its present state to be fit to serve as a global file system. The RZG runs AFS in a WAN environment for

providing software binaries, home directories of the users and space for experimental data for more than eight years now.
Thus, it is recommended to proceed with AFS and stop evaluating AVAKI as a global file system. However, for special use-cases like simplified database-access AVAKI could be considered a possible solution.
Nevertheless, other distributed file systems, like NFSv4[4] and lustre[5] should be evaluated when they appear to be stable. The reason for this is that the installed computer architecture within DEISA is too diverse to be covered by a single GFS, thus alternatives are always welcome. The only condition is that they are supported by enough OSs and vendors which guarantee support and widespread use of it outside DEISA.

# 4. Technical Annex : Evaluation of Avaki Data Grid v. 5.0

## Introduction

The aim of this paper is to present the results of the evaluation of the last Avaki Data Grid v5.0 (ADG5) release. It is the continuation of the first evaluation that was done on the previous version 4 release (cf. [WhitePaper]).

The first part describes briefly the new features included in the v5.0 release. The second part presents the new testing configuration based on temporary mappings and explains the reasons of the configuration changes. The third part describes the details of the testing methodology used for the evaluation and gives its results. In the last part, general observations based on the evaluations results are presented to give an outlook of what can be done with ADG in the DEISA environment.

## 1. New Features

The following new capabilities can be noticed in ADG version 5 :

- Web Service Interface : allows applications (web services clients) to access data through a web service interface (SOAP)
- Web Services : generic web services are included in the product and manage the interactions with all the data sources : files and directories, relational database operations (DB-Ops) and data services.
- New data integration features : a data service extracts data from different data sources : files, other data services, database operation, http operations and produces output in a rowset or XML. Plug-ins can be created automatically with commercial tools such as “Mapforce” from Altova.
- XML Schema Generation & Management : XML schema for any Database Operation can be generated and can be used by Altova Mapforce for the plug-in generation.
- Data Auditing : the new logging facility allows to capture information about how data is being used.
- Load Balancing : Multiple share servers can be associated to one share and the serving work can be distributed among each share servers. Gridservers can be used now in the same manner.
- Temporary mappings : Unix users have the possibility to authenticate themselves on a GDC as an Avaki user and then perform a temporary mapping at the DGAS level between the Avaki and Unix accounts. The mapping expires automatically after a pre-defined time-out.

# 2. Testing Configuration

## 2.1 Motivations for the new testing configuration changes

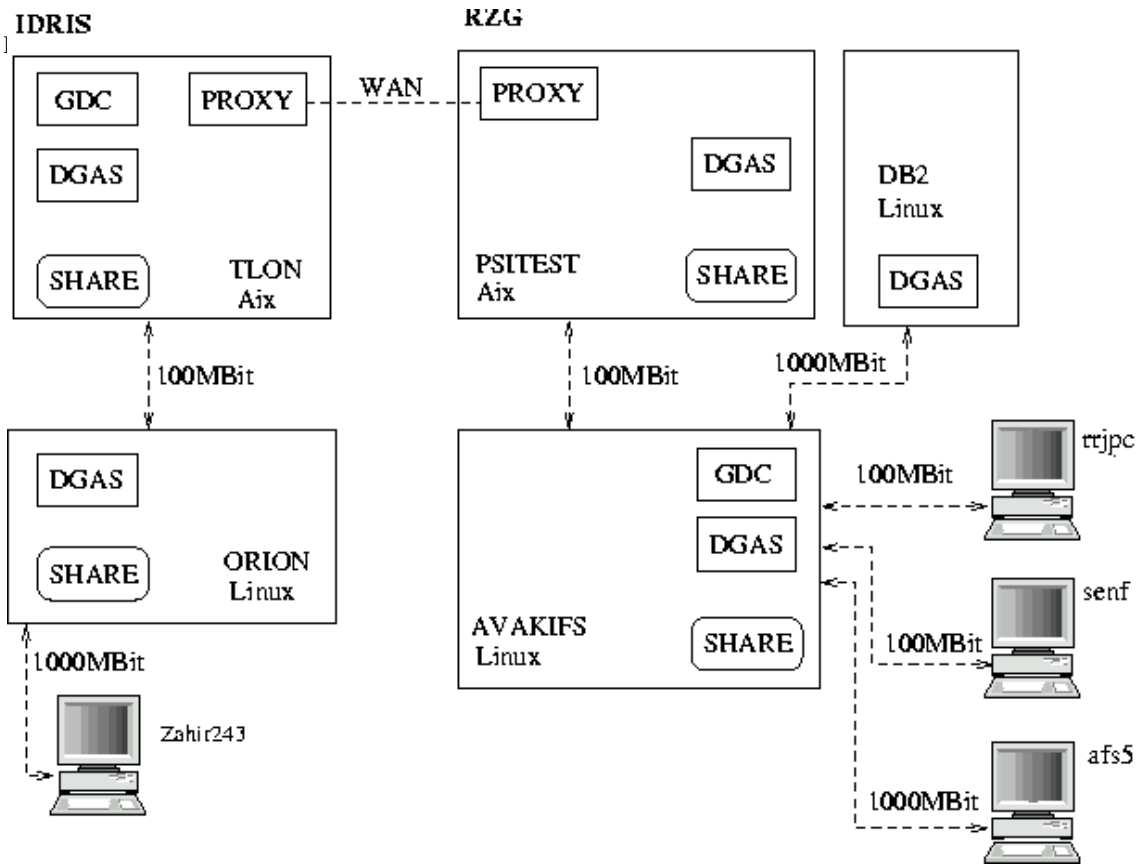
In [WhitePaper] the testing configuration was based on permanent intra- and inter-domain mapping relations between the user accounts at the Unix and Avaki level. The evaluation of this mapping technique revealed two major drawbacks: the first drawback concerns the very high number of important mapping relations to maintain, particularly the inter-domain mappings that are DGAS and host specific. The extension of the number of sites seems unrealistic under these conditions. The second drawback is a security issue that concerns also the inter-domain mappings. Using an inter-domain mapping relation, the remote domain must grant to the local administrator user read and execute permissions on some internal security object. The local administrator domain must also be granted read, write and execute rights on the user directory in the remote domain. This means that the domain administrators must trust each other.

In order to solve these problems, Avaki suggested to use a new feature in ADG5 called "temporary mapping", which is tested below.

## 2.2 Testing Configuration

2.2.1 General configuration

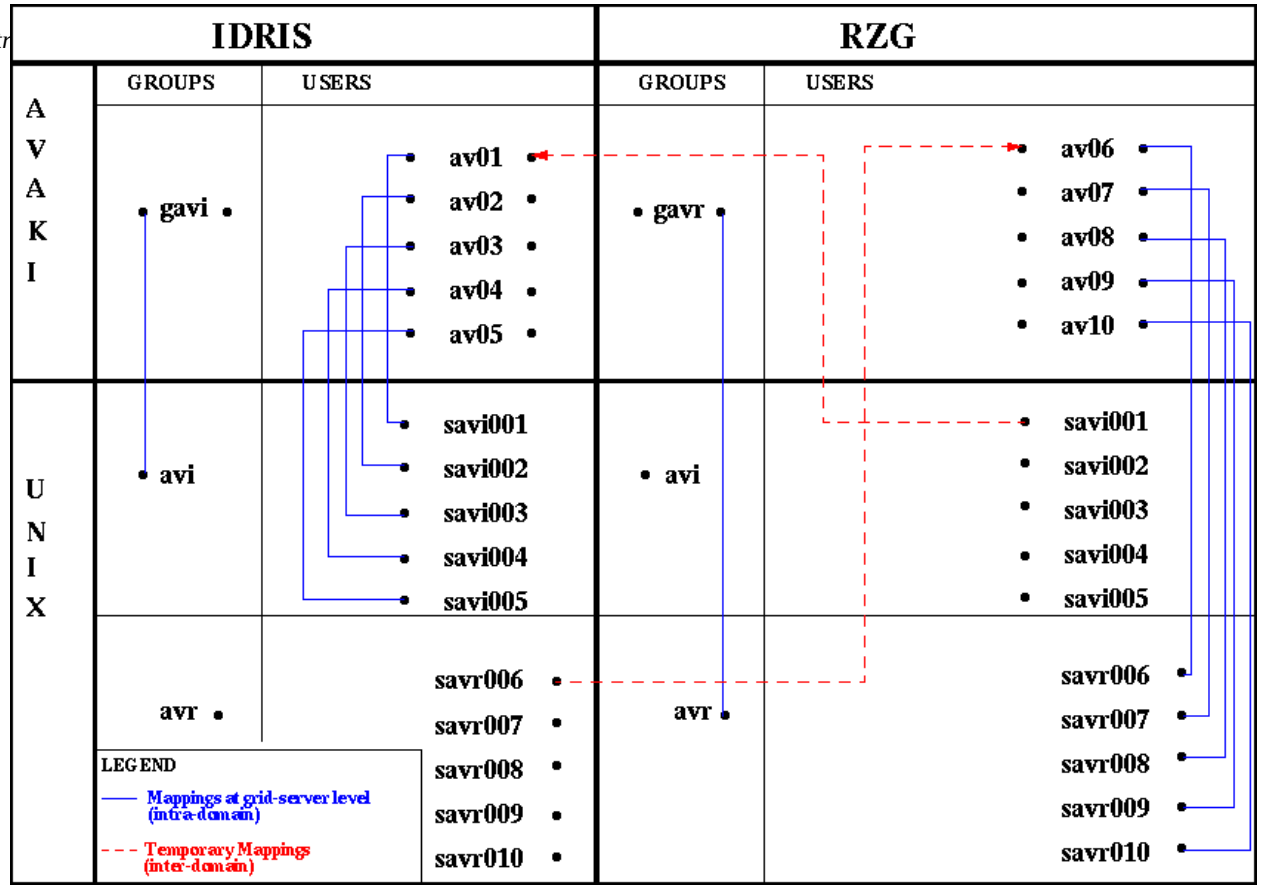
The new testing configuration is represented on the following scheme.




2.2.2 User accounts

The Unix and Avaki account configuration is nearly the same than in the initial testing configuration. As before, both IDRIS and RZG manages 5 Avaki accounts (IDRIS : av01-av05 RZG : av06-av10), two Avaki groups (IDRIS : gavi RZG : gavr), 10 Unix accounts (savi001-savi005 savr006-savr010) and two Unix group (avi, avr). The main difference with the initial configuration concerns the mappings. As explained above, the permanent cross-domain mappings are replaced with temporary cross-domain mappings.

*Illustr*


IDRIS
RZG
GROUPS
USERS
GROUPS
USERS
A V A K I
U N I X
gavi
av01
av02
av03
av04
av05
gavr
av06
av07
av08
av09
av10
avi
savi001
savi002
savi003
savi004
savi005
avi
savi001
savi002
savi003
savi004
savi005
avr
savr006
savr007
savr008
savr009
savr010
avr
savr006
savr007
savr008
savr009
savr010
LEGEND
Mappings at grid-server level (intra-domain)
Temporary Mappings (inter-domain)

# 3. Evaluation

## 3.1 Methodology

We focused our evaluation on four points :

1. Cross-domain temporary user mapping feature,
2. Local performance of the ADG5 as an NFS-exporter,
3. Cross-domain performance of the ADG5 as an NFS-exporter,
4. Local performance of the data transfer via the http-interface and
5. Integration of SQL databases in an Avaki domain.

The features 2. and 3. focus on the use of the ADG5 as a normal NFS-like file system. The features 4. and 5. may be useful for some specific scientific communities, like e-health or bio-informatics, which access their data through some middleware. Feature 1 is useful for granting data-access to users not part of the local domain, thus it is fundamental to the other four use-cases of ADG5.

## 3.2 Functional tests

### 3.2.1 Temporary cross domain mappings

In order to enable cross domain temporary mappings, permissions on several objects in the grid need to be changed by the domain administrators. As an example it is assumed that the Unix user savr006 on orion (IDRIS) wants to perform a temporary mapping to his av06 Avaki account managed on RZG GDC.

First, av06 will need to be able to access the DGAS at IDRIS in order to successfully run the self-map command on the command-line. This user must be given the read access on some (ca. three) system-objects in the IDRIS domain. Furthermore, the IDRIS Administrator needs user read access on a few (ca. seven) system-objects in the RZG domain. Finally, the IDRIS Administrator user needs in IDRIS domain read and execute access on some user-specific object. Contrary to the prior mapping technique, no write access on user-specific object is required.

The configuration required to allow users to perform temporary mapping is still a nontrivial task. The total number of objects, where the special ACLs have to be set, increases quadratically with the number of Avaki-siteslinearly with the total number of the users of all sites . This is feasible with two sites, but as the number of sites increases, the administrative effort becomes overwhelming using only the tools Avaki provided. Since security issues are involved, the development of scripts to do this task and to check the integrity of the system-objects is unavoidable.

Once the permissions are set, a user can activate his temporary mapping using the Avaki command line client. During this procedure, the password of the user has to be typed in, which can be problematic in a batch environment, because the user have to protect their batch-scripts very carefully.

This temporary mapping expires automatically after a time-out which can be set by the Administrator and the user himself. The user can destroy this mapping any time with the

command line client.

Our tests have also shown that simultaneous access with temporary mappings can induce synchronisation problems. If, for example, a user accesses his avaki directory located at RZG from both his Unix IDRIS account (using a temporary mapping) and his RZG Unix account at the same time, he will not always see using his RZG account the files he created from his IDRIS account. Most of the time they appear after having created a new file from his RZG account.

### 3.2.2 Database Access

The integration of a database in an Avaki Data Grid may be an interesting feature for some specific scientific communities, like e.g. bioinformatics, where data is stored in a relational database. The result has to be formatted in a specific format understandable by the user-application. Avaki provides for this a technology that transparently sends a pre-defined query to the database and that transforms its results according to a specific pre-defined transformation, so that the applicaton can understand the output.

In order to test this feature, a MySQL server was installed on the IDRIS side and a very simple database was created on the server. After some more installation issues in the ADG5, it was possible to create a pre-defined SQL query. The output itself can be seen on the web-interface and partly through the NFS-interface. The two standard outputs, which are XML and XSD are not very helpful for users because they are "polluted" by meta information of the database table. ADG5 provides two techniques to solve that problem: view generators and data-services.

#### View Generator

A view generator queries a database and saves the results as a so-called "generated view". A database view generates a file that contains the formatted output of the query either periodically or on demand. The tests have shown that the output file is saved by default in some special directory, which cannot be easily NFS-exported by ADG5. The NFS-support is still unusable at the moment, e.g. reading some generated view does not perform a database-query, but returns the result of the last query.

The output format of the document can be a XML tree, a HTML table, or it can be the result of a pre-defined XSLT transformation. This means that if a job does not understand one of the above data-formats, the user has to develop a transformation suited to his application.

A database operation can take input parameters. This allows the creation of customizable pre-defined queries. *This only works well with the web-gui. If one wants to use the NFS-interface, one has to specify* the input parameters in a XML file and wait for the next query to perform, otherwise he gets the result of the last query, as already stated above. This technique only works if the query has been scheduled to occur periodically and it cannot easily be used by an automated application like a batch-job. Unfortunately, a view-generator always writes to the same output file. In order to secure the consistency of the output file, user can create their own view-generator as long as they are members of the privileged DataProvider group.

Data Service

A data service consists of a data service plug-in (compiled JAR file or XSL style sheet) and of an external interface. Boiled down, it extracts data from different data sources : files, other data services, database operation, http operations and produces output in a rowset or XML. Data services are API driven and produce output that can be obtained only though the API. They are exposed as web services and can be scheduled. This feature has not been tested.

## 3.3 Performance-benchmarks at the RZG

This section presents the benchmark results of the RZG-site of Avaki Data Grid software v5.0 , using iozone[], Bonnie[] and a WWW-browser and compares them to the performance values of a native NFS server. RZG focused its tests on the linux environment (GDC and SHARE on Linux) whereas IDRIS focused its tests on the AIX environment. The AIX tests will be discussed in the next section.

The testbed setup (as shown in section 1) has been slightly modified since the first evaluation process [Whitepaper], particulary on the side of RZG. It now consists of three server machines psitest, avakifs, db2 and three client machines (rrjpc,senf and afs5). The role of each machine is:

- psitest: Running a Proxy, DGAS, and Share service.
- db2: Running a DGAS, otherwise using Avaki-HTTP services from avakifs.
- avakifs: Running the GDC, a DGAS, and Share service.
- rrjpc: Using Avaki-NFS services from avakifs or db2.
- afs5: Using Avaki-NFS services from avakifs or db2.
- senf: Using Avaki-HTTP services from avakifs.

The technical characteristics of the machines in use are given in the table below.

| ***Machine*** | ***OS*** | ***Network-Speed*** | ***Processor*** | ***Proc-Speed*** | ***Memory*** |
|---|---|---|---|---|---|
| psitest | Aix 5.1 | Internal:<br>external: Internet | | | |
| db2 | linux | 1000 Mbit | Xeon | 2.8 GHz | 1.5 GB |
| avakifs | linux | 1000 Mbit | Xeon | 2.8 GHz | 1.5 GB |
| afs5 | linux | 1000 Mbit | Xeon | 2.8 GHz | 1.0 GB |
| rrjpc | linux | 100 Mbit | Pentium 3 | 0.5 Ghz | 128 MB |
| senf | linux | 100 Mbit | Pentium 4 | 1.7 GHz | 256 MB |

*Table 1: Technical details of the machines used at the RZG.*

### 3.3.1 Benchmarks

In order to test the performance of the NFS exporter of ADG5, two standard benchmarks were used: iozone [iozone] and bonnie [bonnie]. These benchmarks have been performed on different server-client configurations to understand the influence of different components in the grid, i.e. network-speed, cpu-power etc.

The following five configurations have been used:

1. rrjpc -- 100Mbit --> avakifs --- DGAS, Share on avakifs, client rrjpc.
2. (a) afs5 -- 1000Mbit --> avakifs – DGAS, Share on avakifs, client afs5. (b) db2 -- 1000Mbit --> avakifs -- DGAS, Share on avakifs, client db2.
3. afs5 + db2 -- 1000Mbit --> avakifs -- DGAS, Share on avakifs, clients afs5 and db2 use simultaneously a directory exported by avakifs.
4. afs5 –- 1000Mbit --> Db2 — 1000Mbit --> avakifs -- DGAS on db2, Share on avakifs, client afs5.

On the clients, the NFS shares were mounted using the following options:

mount -o soft,tcp <NFS-SERVER>:/Shares /local_mount_point

To get an idea of the relative performance, the same tests were run on the same hardware configurations, but the linux kernel nfs-server was used instead of the ADG5. The NFS-server was started with the options :

/home_avaki db2(rw,all_squash,anonuid=1481,anongid=4131,async)

### 3.3.2 Iozone results

iozone version 3.217 was started with the following command-line arguments:

./iozone -s 2G -r 128k -c

The results are shown in the table below.

| ***Test*** | ***Write*** | ***Rewrite*** | ***Read*** | ***reread*** | ***random read*** | ***random write*** | ***back-ward read*** | ***record rewrite*** |
|---|---|---|---|---|---|---|---|---|
| 1.: Avaki | 7887 | 7765 | 5210 | 6747 | 4062 | 5133 | 4251 | 1577 |
| NFS | 9190 | 8975 | 9894 | 0 | 8619 | 9357 | 8127 | 4730 |
| speedup | -14% | -13% | -47% | -34% | -53% | -45% | -47% | -67% |
| 2.: Avaki | 14638 | 14585 | 10278 | 32156 | 8688 | 12609 | 8179 | 3096 |
| NFS | 10152 | 13449 | 51771 | 52288 | 27730 | 11215 | 12999 | 10433 |

| ***Test*** | ***Write*** | ***Rewrite*** | ***Read*** | ***reread*** | ***random read*** | ***random write*** | ***back-ward read*** | ***record rewrite*** |
|---|---|---|---|---|---|---|---|---|
| speedup | 44% | 8% | -80% | -39% | -69% | 12% | -37% | -70% |
| 3.:Avaki | 8741 | 7356 | 4373 | 12291 | 3082 | 8047 | 4041 | 1970 |
| NFS | 4230 | 1092 | 10659 | 12264 | 5782 | 1621 | 3887 | 5210 |
| speed | 106% | 574% | -59% | 0% | -47% | 396% | 4% | -62% |
| 4.: Avaki | 15490 | 15032 | 14980 | 21505 | 9853 | 10366 | 6012 | 3169 |
| NFS | 10152 | 13449 | 51771 | 52288 | 27730 | 11215 | 12999 | 10433 |
| speedup | 53% | 12% | -71% | -58% | -64% | -8% | -54% | -70% |

*Table 2: Test results in kB/s using iozone, the speedup is defined as* $\frac{(Avaki - NFS)}{NFS}$.

3.3.3 Bonnie results

This benchmark was started with the command-line options :

./Bonnie -s 2047

The results are shown in Table 3:

| ***Test*** | *Sequential ouput* | | | | | | *Sequential input* | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | *per char* | | *block* | | *Rewrite* | | *per char* | | *block* | |
| | *kB/s* | *CPU* | *kB/s* | *CPU* | *kB/s* | *CPU* | kB/s | CPU | kB/s | CPU |
| 1.: Avaki | 4200 | 77.3 | 6931 | 7.3 | 2745 | 8.5 | 3864 | 74.2 | 9218 | 10.9 |
| NFS | 5240 | 94.7 | 8756 | 7.9 | 3769 | 7.1 | 4066 | 71.2 | 9344 | 7.6 |
| speedup | -19% | – | -21.00% | -- | -27 | -- | -5% | -- | -1% | -- |
| 2.: Avaki | 14135 | 57 | 14929 | 5.4 | 3326 | 1.7 | 9562 | 35.3 | 33848 | 7.3 |
| NFS | 7950 | 33.4 | 10095 | 4.6 | 9097 | 8.4 | 25529 | 96.2 | 53081 | 12 |
| speedup | 77% | – | 47% | -- | -63% | -- | -62% | -- | -36% | -- |
| 3.:Avaki | 8315 | 34 | 12403 | 4 | 1489 | 1 | 7267 | 28 | 21610 | 4 |
| NFS | 8480 | 35 | 9452 | 4 | 5737 | 4 | 10034 | 39 | 13060 | 3 |

| Test | Sequential ouput | | | | | | Sequential input | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| speedup | -2% | – | 31% | -- | -74% | -- | -28% | -- | 65% | -- |
| 4.: Avaki | 13718 | 58.3 | 16514 | 5.7 | 1962 | 0.6 | 11944 | 45.2 | 21723 | 4.8 |
| NFS | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| speedup wrt 2.) | -3% | – | 11% | -- | -41% | -- | 25% | -- | -36% | -- |

*Table 3: Test results in kB/s using Bonnie, the speedup is defined as* $\frac{(Avaki - NFS)}{NFS}$ *. Test 4 has no equivalent NFS test, so its is compared to the Avaki-speed of test 2.*

3.3.4 Discussion

Basically, there are two conclusions one can draw from the results shown above. The first thing is that for slow machines and low bandwidth, NFS is always better than ADG. The second is, that given high cpu-power and a fast network connection, Avaki performs better in writing files than native NFS. NFS outperforms Avaki for reading files.

## 3.4 Performance-benchmarks at IDRIS

This section presents benchmark results of the IDRIS-site that are AIX oriented for the server part (GDC, SHARE and DGAS). The cross domain access was benchmarked as well.

The technical characteristics of the machines in use are given in the table below.

| Machine | OS | Network-Speed | Processor | Proc-Speed | Memory |
|---|---|---|---|---|---|
| Tlon | Aix 5.1 | 1000 Mbit | 8 Power 3 | | 16 GB |
| Zahir 243 p655+ node | Aix 5.1 | 1000 Mbit | 8 Power 4 | 1,5 GHz | 16 GB |
| orion | linux | 100 Mbit | Pentium III | 860 Mhz | 512 MB |

3.4.1 Benchmarks

The same standard benchmarks were used (iozone and bonnie). The results were also compared to the native nfs performances. Most of the benchmarks and mount options used by RZG were used in IDRIS tests. The main difference is that an extra option was added to the iozone test which is the -e option that includes the flush (fsync and fflush) in

the timing calculations. The performance penalty of this option is minor.

The following four configuration have been benchmarked :

Local domain tests :

5. orion – 100 Mbit -> tlon---DGAS, Share on tlon, client orion
6. zahir243 -- 1000Mbit --> tlon---DGAS, Share on tlon, client tlon.

Cross domain tests :

The cross domain access was also tested. As the network is not yet a dedicated connection, smaller file sizes were used (512MB).

7. orion - 100 Mbit -> tlon ..... psitest – 100Mbit -->avakifs --- DGAS, Share on avakifs, "common" directory accessible by anyone without a temporary cross-domain mapping.
8. orion - 100 Mbit -> tlon ..... psitest – 100Mbit -->avakifs --- DGAS, Share on avakifs, directory on which a temporary mapping is required.

3.4.2 Iozone results

The results are shown in the table below.

Options used : iozone -i0 -i1 -i2 -i3 -i4 -r128k -s2g -c -e

| *Test* | *Write* | *Re-write* | *Read* | *Re-read* | *Random read* | *Random write* | *Back-ward read* | *Record rewrite* |
|---|---|---|---|---|---|---|---|---|
| 5.: Avaki | 5919 | 6587 | 6187 | 2857 | 1543 | 3618 | 1617 | 3045 |
| NFS | 1635 | 7932 | 5693 | 5736 | 6906 | 686 | 5860 | 7546 |
| **Speedup** | 262% | -17% | 9% | -50% | -78% | 427% | -72% | -60% |
| 6.: Avaki | 14088 | 16436 | 8654 | 15220 | 13060 | 4795 | 592 | 612 |
| NFS | 10959 | 13826 | 16080 | 25610 | 85401 | 7099 | 57707 | 17129 |
| **Speedup** | 29% | 19% | -46% | -41% | -85% | -32% | -99% | -96% |
| 7.:Avaki | 1550 | 1613 | 4794 | 4948 | 12050 | 2737 | 1351 | 1261 |
| 8.: Avaki | 1730 | 1762 | 4610 | 10725 | 3494 | 2989 | 1255 | 1255 |

*Table 6: Test results in kB/s using iozone, the speed up is defined as* $\frac{|Avaki - NFS|}{NFS}$ *. Tests 7 and 8*

*have no equivalent NFS test.*

### 3.4.3 Bonnie results

This benchmark was started with the command-line options :

./Bonnie -s 2047

The results are shown in the table below :

| ***Test*** | *Sequential ouput* | | | | | | *Sequential input* | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | *per char* | | *block* | | *Rewrite* | | *per char* | | *block* | |
| | *kB/s* | *CPU* | *kB/s* | *CPU* | *kB/s* | *CPU* | kB/s | CPU | kB/s | CPU |
| 5.: Avaki | 5578 | 63.8 | 6650 | 6.3 | 400 | 0.4 | 2596 | 29.6 | 2359 | 0.7 |
| NFS | 1334 | 15.1 | 1765 | 1.3 | 2378 | 3.2 | 5158 | 59.1 | 5627 | 3.2 |
| **Speedup** | 318% | 323% | 277% | 385% | -83% | -88% | -50% | -50% | -58% | -78% |
| 6.: Avaki | 14438 | 19.1 | 14292 | 6.6 | 2648 | 1.7 | 6613 | 9 | 436 | 0.1 |
| NFS | 12401 | 16.2 | 12296 | 5.8 | 4173 | 4.2 | 23273 | 27.6 | 68110 | 9.8 |
| **Speedup** | 16% | 18% | 16% | 14% | -37% | -60% | -72% | -67% | -99% | -99% |
| 7.:Avaki | 1299 | 14.8 | 1584 | 1.3 | 238 | 0.3 | 2400 | 27.1 | 1030 | 0.3 |
| 8.: Avaki | 1286 | 14.6 | 1563 | 1.2 | 215 | 0.3 | 4541 | 54.7 | 10944 | 3.5 |

*Table 4: Test results in kB/s using Bonnie, the speedup is defined as* $\frac{|Avaki - NFS|}{NFS}$ *. Tests 7 and 8 have no equivalent NFS test.*

### 3.4.3 Discussion

The results are very similar to RZG's, performances seem to be comparable on both Linux and AIX environment.

The DGAS access is significantly more performant than the NFS mode for the write transfers but all the other transfers are most of the time less performant than the NFS mode. The test number 5 gives a very high speed-up for the write mode because the native NFS performances were unexpectedly low, probably due to a network configuration problem.

Concerning the cross domain tests, both test7 and test8 gave similar results. Thus, the use of a cross domain temporary mapping doesn't affect the performance, as expected. The cross domain performances seem acceptable given that the two sites are not connected with a dedicated fast connection. To give an idea of the current connection speed, several scp transfers were performed between the two sites and the rates varied from 680 KB/s to 2500KB/s.

## Webinterface

Beside the NFS-tests, the performance of Avaki as a file sharing utility via the http-protocol as it is used by web-browsers has been tested.

To keep the tests as simple and reproducible as possible, the "ui-server" of the ADG 5.0 which gives only rudimentary file-sharing possibilities has been used. The benchmark itself examines only the speed of the connection by transferring a file from the local file system of a client into the datagrid. The integrity of the data has not been tested. Two different clients with a 100MBit and a 1GBit connection to the share-server have been used to examine the influence of the network speed on the overall performance.

The main characteristics of the machines involved in that test are given in the table above.

The tests consist of a simple read and write of a file from and to the share-server.

1. Reading/Writing of a file by client senf only.
2. Reading/Writing of a file by client db2 only.
3. Reading of the same file by client senf and db2 simultaneously.
4. Reading/Writing of two different files by client senf and db2 simultaneously, i.e. first both clients read, then both clients write.

These four tests were performed using the filesizes of 128MB and 1.5 GB respectively. It was not possible to use files of 2GB length, which are still in the 32-bit range, because the webrowser would not understand them. The web-browser just showed a file length of 0 byte. The two different file-sizes were used to see caching-effects of the local memory. Since there is no time-measuring tool in the ui-server, the transaction-time was measured by scripts which watch the target directories. The error made by this procedure, first start the transaction in the web-interface, then start the script is taken to be 1 sec., which is almost irrelevant for the measurements. Each of the test was done three times to get an estimate of the standard deviation of the results.

| ***Test #*** | ***1: senf*** | | ***2:db2*** | |
|---|---|---|---|---|
| | read | write | read | write |
| Filesize / MB | 128 | 128 | 128 | 128 |
| Time / s | 73±2 | 184±4 | 16±2 | 164±5 |
| Throughput / MB/s | 1.75±0.05 | 0.7±0.02 | 8±1 | 0.78±0.02 |
| Ratio db2/senf: | 4.6 | 1.1 | | |
| Filesize / MB | 1536 | 1536 | 1536 | 1536 |
| Time / s | 757±23 | 2213±74 | 499±12 | 2036±7 |

| ***Test #*** | ***1: senf*** | | ***2:db2*** | |
|---|---|---|---|---|
| Throughput / MB/s | 2.03±0.06 | 0.69±0.02 | 3.08±0.07 | 0.75±0.005 |
| Ratio db2/senf: | 1.5 | 1.1 | | |
| Ratio 128MB/1536MB= | 0.9 | | 2.6 | |

**Discussion of the singular file tranfers:**

Beside the fact that the actual performance values are poor, one can learn two things from the table above. Comparing the two speeds achieved by the same clients it shows up that the reading-speed is cache-dependend, the writing speed is not. The reading speed of "senf" is actually higher with the large volume. This is believed to be irrelevant, because the client is also used as a normal office-machine and thus the alien load can differ considerably. The only thing which can be said is that the memory of senf is too small to cache the file effectively when being used as a normal office-computer (i.e. running a window-manager, the web-browser and so on). The reading speed of the client "db2" of large files is much lower (by a factor of 2.6) than with small ones. Here, the client has enough memory to cache the small file effectively in the memory, but not the large one. The writing speed of the clients is independent on the file size. The web-browser apparently just pushes the data to the server, which deals with it. The performance of the server is apparently so slow that the writing speed on a 1GBit network is barely faster than on an 100 MBit one.

Comparing the two clients for both filesizes gives information about the influence of the network-speed on the performance of reading from the AvakiDG. The writing-performance is the same so no further conclusions can be drawn. Looking at the small file, the reading-speed of db2 is much higher, because of the cache-effect discussed above. The performance of the client db2 of reading a large file from the AvakiDG is just 1.5 times better than the performance of the client senf. This is disappointing considering the fact that the network speed is 10 times higher. Apparently the capabilities of the server can be almost exhausted by a 100 MBit line.

**Simultaneous access to the Avaki-server:**

The upper test-case has nothing to do with real life and was just used to get some basic properties. In this section, the two clients access simultaneously on the avaki-server in order to estimate it scaling capabilities. This is still far from a real-life situation, but it is technically difficult to go to much higher numbers which would be reached in the DEISA-project.

| *Test#* | *3* | | *4* | | | |
|---|---|---|---|---|---|---|
| | *senf* | *db2* | senf | | db2 | |
| | read | read | read | write | read | write |

| *Test#* | *3* | | *4* | | | |
|---|---|---|---|---|---|---|
| Filesize / MB | 128 | 128 | 128 | 128 | 128 | 128 |
| Time / s<br><br>Through-put / MB/s | 79±3<br><br>1.62±0.06 | 15±1<br><br>8.53±0.57 | 55±5<br><br>2.33±0.21 | 189±3<br><br>0.68±0.01 | 28±5<br><br>4.57±0.82 | 170±1<br><br>0.75±0.005 |
| Ratio db2/senf: | 5.3 | | 2 | 1.1 | | |
| Filesize / MB | 1536 | 1536 | 1536 | 1536 | 1536 | 1536 |
| Time / s | 643±11 | 489±12 | 761±16 | 2407±65 | 496±10 | 2119±30 |
| Through-put / MB/s | 2.39±0.04 | 3.14±0.08 | 2.02±0.04 | 0.64±0.02 | 3.1±0.04 | 0.72±0.01 |
| Ratio db2/senf: | 1.3 | | 1.5 | 1.1 | | |
| Ratio 128MB/ 1536MB= | 0.7 | 2.7 | 1.2 | 1.1 | 1.5 | 1 |

The reading of the same file takes roughly the same time on both clients as it takes for reading the files separately. This is true for both filesizes. Thus, one can say that AvakiDG takes no advantage of the fact that the same file is accessed by different clients. This is not suprising for the 1.5 GByte file, but the smaller file fits easily in the memory of the server which has no other load other than this test, so a caching was possible. On the other hand, the performance did not degrade significantly. Apparently, the server can serve multiple streams at the same time with the same speed as it serves a single stream. This behaviour can be seen in all the results (with the exception of reading a 128 MB file over a GBit line), are basically the same as in the tests with a single stream. A plausible explanation is that the avaki-server is threaded and its performance is such that two threads do not fill its 1GBit network nor does it have any effect on the CPU-power of the machine. The latter was to be expected, since the avaki-server machine has two cpus. The first point is not surprising as well, since even a single data-stream when reading a file filled its line only by 2% of its theoretical value.

# 4. General Observations

## 4.1 Comments on the technical feasibility of the tests.

In general, it can be said that for the correct setup of Avaki Grid software a plenty of know-how is needed to avoid common pitfalls.

While testing the Avaki Data Grid, following observations are made while testing the different issues.

Temporary Cross-domain mapping:

Many ACLs still have to be manipulated in order to get the user-mapping to work. If a single ACL is not set right, the whole procedure fails without any specific error-message.

NFS-tests

- NFS clients must use the 'tcp' switch at the 'mount' command otherwise data transfer will stop randomly with 'write: Input/output error' messages.
- If NFS clients use the 'actimeo=0,noac' switch at the 'mount' command performance will affect badly. [Vincent Ribaillier, IDRIS].
- Logging at level 'DEBUG' must not be enabled on any of the components of Avaki. [Greg Haber, Avaki Support]
- Occurs randomly: if user 'root' mounts an Avaki-NFS share, a normal unix user (savr006) could not access its directory unless root has visited the directory. savr006 gets ``No such file or directory'' unless root has been there. Seen on RZG site only.
- System is not usable if local file system is full
- At RZG, DGAS could not be started if cache was lost (e.g. deleted on file system level). [solution: remove whole DGAS/cache and DGAS/dgas_db directory]. At IDRIS, the DGAS found itself several times in an inconsistent state. The solution was also to proceed to a complete deletion of the cache and of the cache data-base.

Webapplications

Beside the fact that the operating system is caching files in memory, a file-caching on the hard-drive takes place as well. When one wants to upload a local file to the AvakiDG, the ui-server first copies it to a temporary-directory and then across the network to the real Avaki server. This slows down the upload considerably if the network speed is reasonably fast. A further nuisance is that this "cached" file is not used again when one tries to upload the same file a second time. On the contrary, it writes a second tmp-file in the avaki-ui-server-directory. This fills up the file system on which this directory is residing, thus rendering the machine unusable for AvakiDG and possibly for everything else if the ui-server is on the root-file system. For the tests, the author had to delete these "cache"-files manually between two tests, otherwise his machine would become unusable. When downloading a file from avakifs to the local disc-space, the browser mozilla first writes it in the directory /tmp and then to the target directory, thus causing

an overhead on its own. Other web-browsers like "konqueror" from the KDE-project write the temporary file into the target-directory. The last nuisance is connected to the file-caching while writing into the AvakiDG. For uploading, there is a check-option if ones wants to overwrite a previously existing file. If this is unchecked and the file exists, then the upload is refused. Unfortunately, this check is performed after the file is transferred into the temporary directory, thus this refusal may come only after several minutes of waiting in vain.

Other problems

Beside the issues pointed out above, two more general problems came up:

- No support for global quotas

- Data Backup and recovery mechanisms. No acceptable technique was find to save and restore files with the guarantee that all the original meta-data (permissions and ownership) are well recovered on the three different views.

## 4.2 Conclusion and consequences for Deisa

Overall, it has to be said that AvakiDG is not mature enough for heavy use. Although it has become better since the last release, there are many pitfalls which are not covered by the manuals. Another point is that the three interfaces (command-line, nfs and web) are still entangled so that a user has to learn all of them. Thus the ADG seems unsuitable for a batch environment where the jobs have a complex structure on their own.

The usage of the AvakiDG via the http-protocol is too slow for large amount of data.

We focused our evaluation on the temporary mappings, the database access and the performances.

# Bibliography

[White Paper] *Avaki Data Grid White Paper*, Vincent Ribaillier, 13/01/2004
[iozone] www.iozone.org, used version: v3.217
[bonnie] www.textuality.com/bonnie