



# Arch White Paper

## 1. Introduction

This document is written for webmasters and IT professionals who are interested in high performance search engines. It may also be of interest to business executives who are seeking a better alternative to the search engine in use at their organisation.

Arch is an extension of the Apache Nutch, designed for efficient and effective indexing and search of organisational web sites (intranets). The corporate environment has a few very distinct characteristics as opposed to the global Web, for which Nutch was originally designed. Obviously, requirements to scalability are sufficiently lower when indexing an intranet than when indexing the WWW. On the other hand, requirements to security are higher. Below is a list of the distinct requirements of intranets compared to those of the WWW:

- Document-level security. No search should reveal existence of documents to users who are not supposed to know about them. Administrators must be able to define who can find what.
- Inexpensive index updates. Frequent whole intranet re-crawls are out of question. This would make the crawler a major (ab)user of corporate web servers, putting a high load on them.
- High availability. The search engine should be available 24/7, because document search and retrieval are an important part of decision making processes.
- Supporting multiple web sites should be easy, as companies often have more than one web site. Adding and removing a site must be easy, and they must be reasonably independent to administer.
- Link based document weighting algorithms (like Google PageRank) do not work as well on intranets as they do on the WWW. This is because a) links on intranets reflect more site structure than page popularity and b) there are just not enough redundant links to estimate page quality. Redundancy is avoided when possible because it increases maintenance costs. An adequate document weighting algorithm is required.
- Web search companies normally have a dedicated team of professionals with required skills to accomplish the task. With the exception of large enterprises, intranet search engines are deployed and maintained by a webmaster or a member of IT staff, and this is not their full-time occupation. These people usually have a limited, if at all, Java experience. But, normally, they have experience in the Apache-PHP-MySQL trio and able to deploy applications like Tomcat. An intranet search engine should be designed in such a way as to allow people with this skill set deploy, customize and maintain it with minimal efforts.
- Similarly, much smaller hardware resources will be allocated to indexing the intranet. Often, it will be just one server.

On the positive side, there is more detailed expert knowledge available when indexing the intranet, than when indexing the WWW. Web server logs are also available and are an excellent source of information.

This document consists of an introduction and two parts. The first part, Features Overview, presents a set of Arch solutions addressing the challenges of intranet search listed above. The second part, Architecture Overview, reveals more implementation details “under the hood” and

hopefully provides enough information on where to dig in the source code if the reader wants to change or add something.

This document is one of the three (the other two are the deployment manual and sample configuration file) that are recommended to get familiar with before deploying Arch.

## 2. Features Overview

### 2.1 Improved Document Weighting

Google won the search engines battle of the 90's mainly because they use the "quality" component when computing the results scores. This component is estimated based on the information contained in the Web links graph. Simplistically, the more links point to the page, the higher quality it is estimated to be. Links leading from higher quality pages have higher weight when counting the scores of the pages they point to. Google has a patented algorithm allowing it to compute scores of pages on the web and neutralize attempts of spammers to artificially manipulate these scores. Using these scores, the Google engine tends to find "better" pages.

However, this method can't be used for intranets, for the lack of such information. While links on the Web can be used for estimation of popularity of a page, intranet links mostly reflect the site structure. Having multiple links to the same page means redundancy, is harder to support and is avoided when possible. For example, the CASS MIRIAD software home page has 26 external Web links pointing to it and only 2 internal links. There is no way to detect that this is a major page based on the internal link count. Some significantly less important pages have more links pointing at them.

Web logs are normally available in the intranet environment and can be used for estimation of document quality based on document access frequency instead of, or in combination with, link counts.

Arch has a log processor module that goes through available logs before indexing, and updates document weights in a relational database. For each document, assigned weight is proportional to the logarithm of access count in the period covered by the logs. There are a few rules used to identify and filter out noise, such as log records generated by a search engine crawler accesses.

Switching to this method of quality estimation made a dramatic effect. The MIRIAD home page moved instantly to the top of the first results page when searching for MIRIAD. Other major pages are also easily found. In short, now people find what they expect to find. The difference is comparable to that between Google and pre-Google search engines.

But, log processing makes more things possible:

- It finds new pages without the need to recrawl;
- It finds isolated pages;
- It can help to index on-line databases (we have not done anything in this area yet).

Thus, isolated pages and new pages are indexed automatically by Arch on the day they are found in logs.

It should be mentioned that it is not necessary to process all available logs to derive reasonable document weights. Popular documents will appear in a relatively small sample. The rest can be safely considered equally "unpopular" and assigned a minimal weight. If this optimization is used, there is a risk to miss new URLs and isolated pages that may not appear in the selected log sample. However, if this information is not critical, the cost of document weight calculations can be significantly reduced.

### 2.2 Document Level Security

As noted above, document security is a must in the corporate environment. Arch provides a couple of ways to assign user and group names to a document or a folder tree. If a document or

a folder does not have anything assigned explicitly, it inherits these attributes from the folder containing it. The assigned user and group names are indexed together with the documents in the “ar\_user” and “ar\_group” fields. When a user performs a search, the user’s name and groups are automatically added as a filter to the query and only documents that have this user’s or group’s names associated with them are found.

### **2.3 High Availability**

Starting with version 1.4 (based on Nutch 1.4), Arch uses Apache Solr as its index/search server. Solr is a high quality, very flexible and scalable application, with proven excellent track record. It is more than adequate Arch needs.

### **2.4 Indexing and Search Partitioning**

Not all web sites are born equal. But, even in one site, usually there are areas which are very different in size and patterns of use. Some areas are huge, but virtually static, such as technical documentation. Some are rapidly changing and need to be indexed daily, but relatively small, such as announcements pages and blogs.

It does not make much sense to re-index the whole site often to keep indexes of the rapidly changing pages up to date. It is much better to avoid re-indexing static parts and instead do low-cost frequent re-indexing of the rapidly changing parts.

This is exactly what Arch does. It allows partitioning a site into areas and then assigning to each area its own indexing interval. It also allows specifying on which week days re-indexing can be performed.

The other aspect in which partitioning helps is search focusing. For example, a corporate site can be partitioned into areas such as “finances”, “legal”, “IT”, “HR”, etc. Most searches will benefit from restricting them to a particular area because this will reduce the number of irrelevant hits, even for poorly formulated queries.

### **2.5 Low Cost Index Updates**

With site partitioning in place, reducing the indexing cost is easy: only areas requiring re-indexing are re-indexed. Index data of other areas remains unchanged. This makes the daily crawling of critical areas quite affordable, provided they are of reasonable size.

The other source of updates is logs processing. When a new URL is found in logs, it is automatically indexed.

### **2.6 Multiple Web Sites**

Arch supports multiple web site indexing. Its data folder contains a main (root) configuration file and site folders; one per site. Each site folder contains a logs folder and a site specific configuration file. Adding a site is as easy as adding a site folder with a configuration file. Arch will pick it up when indexing is started. To remove a site it is enough to remove the site’s folder and run a Solr query that will delete this site entries from the index.

This setup is designed to make it possible to provide a search service to multiple sites so that they can be administered independently.

### **2.7 Site Directory**

In the process of log analysis and later, indexing, Arch creates a record for each file and folder in a relational database. This is where document weights and permissions are maintained before incorporating them into Nutch indexes. It also allows maps to be automatically built of site contents or site directories. These directories are then presented as dynamic folder trees and can be browsed by users. Users can explore the site structure and, if authorized, can view and change nodes properties. This is one of the two ways to define security information for site documents and folders (the other is placing an explicit statement in site configuration file).

To see nodes, users have to have at least read permissions. To see node properties, users have to have at least write permissions. These will allow them to change node label and description. To change access permissions, a user has to be listed as an owner of the node.

Any changes will take effect on directory browsing immediately, but they will not take any effect on searches until the changed node is re-indexed because it is the indexing process that places security information in the Arch index.

## **2.8 Lucene Query Syntax**

Solr supports powerful Lucene query syntax, with a few minor differences.

## **2.9 Dual Interface**

Arch includes a light weight PHP front-end that can be used instead of Solr interface. Either (or both) of them can be used for deployment and customization. This significantly reduces requirements to skill set of the personnel deploying Arch. Most webmasters have PHP skills, but significantly fewer of them have the Java skills necessary to understand and customize a fairly complex software package, such as Nutch.

A PHP front-end is a light weight PHP application implementing the whole set of user interface functions and user authentication. It serves as a relay between the user and the Arch Java part and is designed for easy customization.

Except for reducing requirements to staff skills, adding PHP front-ends has also significantly increased the flexibility of the whole system. There is no limit on the number of front-ends served by a single Arch engine. Each of them can be configured to represent a particular set of users and user groups, limited to search a particular site or area, and/or to use its own authentication method.

## **2.10 End User Customization**

There is a set of parameters that each user can customize for themselves. These include visual themes, number of hits per page, default site to search and saved queries.

Visual themes control the look and feel of pages generated by Arch.

The saved queries are a kind of bookmarks that are placed on the query page upon user's request. Clicking on them has the effect of repeating the query. This is not something that one can't achieve with standard browser bookmarks, some people just prefer to go through familiar links to get to a desired page, rather than bookmark the final page and then use the bookmark. If they prefer browsing, why not help?

All end user customizable parameters are individual for each user and kept in browser cookies.

# **3 Architecture Overview**

## **3.1 Software Architecture**

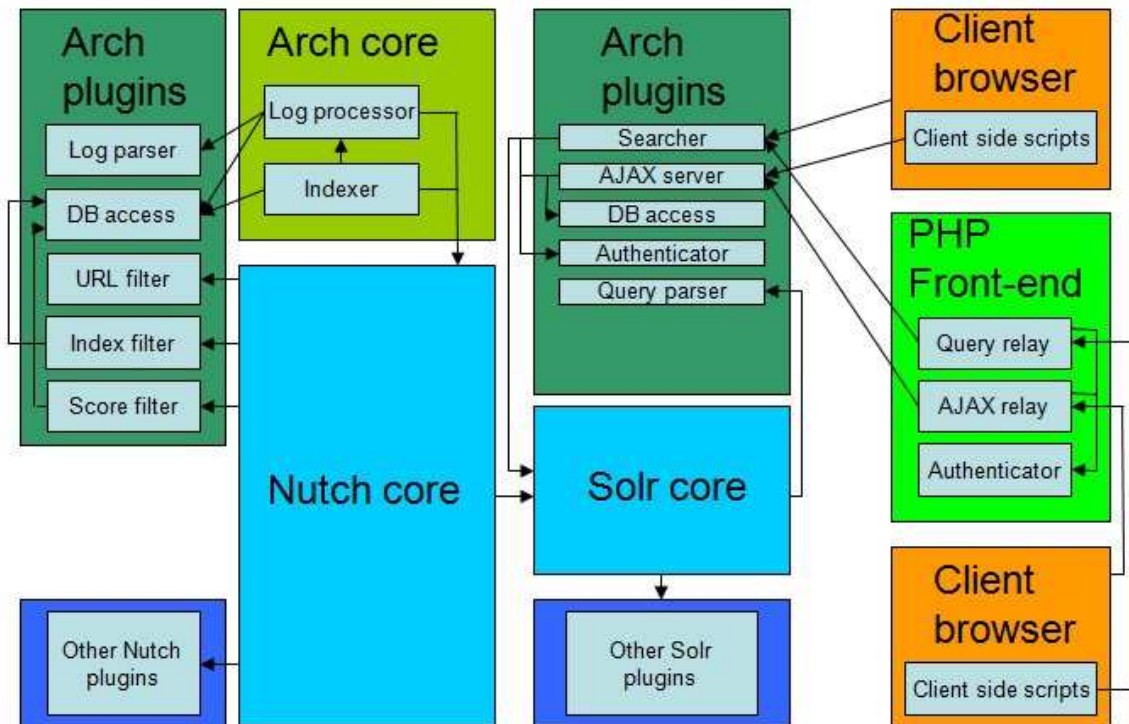


Figure 1. Arch architecture diagram

Figure 1 presents a high level diagram of Arch architecture. Below is a list of main modules with short descriptions.

### 3.2 Arch Core

**Log processor** is implemented by the `au.csiro.cass.arch.logProcessing` package. The main class is `LogProcessor`. It goes through site's logs directories, finds new/changed log files and processes them to calculate document weights and find new URLs for indexing.

It takes configuration parameters from the root configuration file and the site's configuration files. It maintains a list of processed logs so that no reprocessing is done for old logs. There are no requirements for log names. Everything found in the log directories is treated as a log file. Files that contain no log information are ignored.

Before calculating document counts, a list of ignored IP addresses is updated, unless this feature is switched off. An IP address is placed on this list if requests coming from it exhibit a pattern of behaviour common to web crawlers. A good indicator is the number of documents accessed in a short period of time. If this number is above a certain threshold, the IP address gets blacklisted.

When the log processor finds a new document, it creates a database record for the document and records for all the folders in the path from the site root to this document.

To optimize performance, the log processor uses a document cache when counting scores. This cache is periodically freed by saving the least frequent documents' data to the database.

The log processor depends on a couple of plugins. The first one is a log parser. It is responsible for parsing a log line. The default log parser that comes with Arch works with the combined log format. Other formats will likely require a different log parser.

The second plugin is the DB access plugin. It implements an interface to an RDBMS. The default Arch DB access plugin uses MySQL.

**Indexer** is the main class responsible for building an Arch index. It is implemented by the `au.csiro.cass.arch.index` package. If started without the name of a site to index, the indexer goes through the Arch sites folder and attempts to index all sites found there. For each site, it runs the log processor to update document weights.

Then it iterates through areas of each site and checks if the area is due to be re-indexed. If yes, it uses Nutch classes to index the area.

Each area is defined by 3 sets of URLs in site configuration file. The first set is roots. These URLs are used to start the crawling process. The second set is includes. These are URL prefixes that define URLs belonging to the area. The third set is excludes. These are URL prefixes that define URLs excluded from the area. A URL belongs to the area if it is either a root or matches at least one of the includes and none of the excludes.

When Nutch performs crawling, Arch URL filter uses this information to insure that only URLs belonging to the area in processing are crawled.

At the end of indexing, the indexer creates a combined index from the indexes of all the areas and switches the Nutch searcher to this index.

**AJAX server** is responsible for serving requests of client side scripts (JavaScript) and PHP front-ends. It is implemented by `au.csiro.cass.arch.solr.AJAXServerComponent` and `au.csiro.cass.arch.solr.AJAXResponseWriter` classes.

**Searcher** is implemented by `au.csiro.cass.arch.solr.ArchQueryComponent` and `au.csiro.cass.arch.solr.ArchResponseWriter` classes.

### 3.3 Arch Plugins

**Log parser** parses log lines in a particular log format.

**DB access** implements DB access. There are quite a few functions, but they don't use any exotic features, so, porting from MySQL to other SQL flavours should be easy. In fact, it took just one day to one of Arch users to get Arch use PostgreSQL instead of MySQL. Note that Arch is designed to make it possible for each site to store its information in its own database. This made the system a bit more complex than it could have been, but also has added more flexibility.

**Authenticator** is used to authenticate clients and PHP front-ends. See details below, in section 3.10. The default Arch authenticator (`csiro.cass.arch.solr.Authenticator`) implements the Apache file based authentication scheme as well as allowing or blocking access based on configurable IP addresses patterns. However, it is only useful for relatively small organisations and is provided mostly as a reference implementation to help develop customer specific authenticators.

**Index filter** – this filter performs two important tasks. It is called by Nutch every time a document is being added to the index. The Arch index filter retrieves the document's security information from the database and adds this information and site and area names to the document. It also ensures that a record of this document and all its enclosing folders exist in the database.

**URL filter** – as mentioned above, this plugin is responsible for limiting crawling to a particular area. It is implemented by `au.csiro.cass.arch.filters.PrefixURLFilter`.

**Scoring filter** – takes document weights from the db and uses them to assign Nutch scores. It is implemented by `au.csiro.cass.arch.filters.Scoring`.

### 3.4 PHP Front-end

The PHP front-end was added to Arch with the primary intention of making deployment and customization easy for people who have PHP skills. The 3 major things that need to be customized are appearance, authentication and language. All of them can be easily customized in the PHP part of Arch. Webmasters who prefer PHP don't have to do anything with Java or JSP. The PHP front-end uses the Smarty template engine to form output pages. Smarty is widely used and easy to learn for those that are not familiar with it. All output templates and messages visible to end users are located in a single directory under "language code" for easy translation to the required language.

A simple authentication module implementing the Apache file based authentication scheme is included for use in organisations where this authentication method is used and as reference code to assist development of custom solutions.

Please note that in the used authentication implementation PHP front-ends send the front-end ID and password with each request as clear text. This means that there is a possibility of a middleman attack and the connection between Arch server and PHP front-end must be secured.

**Note:** when customizing PHP, please place changed files in the `local` subdirectory of the directory where the original file is. The system always checks (with exception of `localname.php`) whether a local copy of the file is available and if it is, it uses it. This is done to prevent your changes from being overwritten when upgrading to a newer version.

**Query relay** – serves query forms, takes submitted requests. After receiving a request, it adds the user and group names for authenticated users, or “guest” and “public” for unauthenticated users, front-end ID and password, and sends the augmented request to the Arch server. After receiving results, it uses Smarty templates to form a results page and sends it to the client.

**AJAX relay** – receives AJAX requests from client side scripts, adds user and group names, front-end ID and password, and sends the augmented request to Arch AJAX server. After receiving results, uses Smarty templates to form HTML code if needed and sends it to the client.

**Authenticator** – a PHP analog of the authentication plugin, except it does not have to implement authentication of front-ends; only client authentication. It has a similar interface. A reference implementation is provided.

### 3.5 Client Side Scripts

These scripts implement the functions necessary to display dynamic pages, such as the site directory. They use AJAX to request required data either from the Arch server or from a PHP front-end.

### 3.6 Arch Configuration Location and Structure

Arch expects that Arch configuration directory is in `$NUTCH_HOME/conf/arch`, where `$NUTCH_HOME` is the value of the environment variable that Nutch requires to be set. Arch expects that its data directory contains a global (root) configuration file named `config.txt` and `sites` folder. In turn, the `sites` folder should contain site folders with names corresponding to sites' names. Each site folder must contain a site configuration file named `config.txt` and, optionally, a `logs` folder where logs used for document weighting are placed.

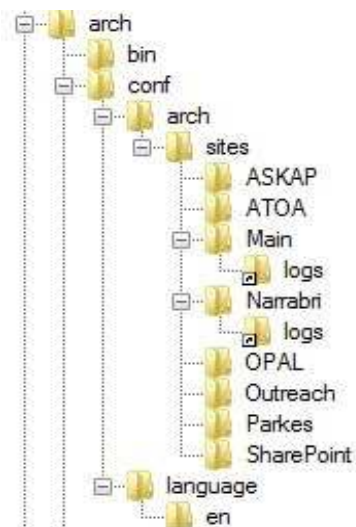


Figure 2. Arch configuration directory example

An example of Arch configuration directory is in Figure 2. Here, `arch/conf/arch` is the Arch configuration directory, containing a “sites” folder with a folder per site. `ASKAP`, `ATOA`, `Main` etc. are site names.

The `language` directory contains language specific folders, each containing Velocity templates translated into that language.

### 3.7 Arch Configuration Format

Arch keeps configuration parameters in textual files similar to Java properties files. Each line is expected to be a key-value pair, except if the line is empty or starts with ``#'` or ``//'`. Then it is a comment. Keys and values are separated by the `'='` character. The first such character in each line is taken as end of the key. If there are more in the same line, they are included in the value.

Some parameters in the site's configuration files can override values defined in the root file. For example, the root file must contain a db address and access parameters. These values can be overridden in the site configuration file to use a different database for the site's data.

Area-specific parameters are specified by adding the area name at the end. For example:

`area = news` declares that “news” is an area name.

`enabled.news = on` states that this area is enabled.

`depth.news = 10` sets crawling depth for area news to 10 iterations.

### 3.8 Databases

Here the RDB tables used by Arch are described. Exact specifications can be obtained by examining the Arch db access plugin or by listing table descriptions in the database after running the Arch indexer.

**Areas** table contains a list of all areas with the status of each area and date of last recrawl.

**Logs** table contains a list of all log files. For each file, it contains information on date range covered and the last modification time.

**Roots** table contains the URLs used in area definitions: roots, inclusions and exclusions. If any of these are changed, the area becomes due to recrawl.

**Sites** table contains information about sites: the date range which was used to calculate document counts and weights, and the site root node parameters used in site directory.

**Site\_<site name>** tables are created for each site and create a linked tree presentation of site's structure, where nodes correspond to documents and folders. Each node contains permissions information, presentation information, document weight and links to the next node on the tree level and parent node.

### 3.9 Search

When receiving a search request, Arch search module uses the authentication plugin to establish identity of the user. If there is no authentication information in the request or session data, the user is assigned the name “guest” and the group “public” (however, this depends on the used authentication plugin). The search is always limited to documents “visible” to the user. To ensure this, the search module adds user and groups names to the query as a filter. PHP front-ends also get authenticated and can be limited in which users and groups they can represent and which sites they can search. This all depends on the used authentication plugin. See details in the next section.

### 3.10 Authentication

The purpose of the authentication process in Arch is to determine the filters that restrict what user can find using Arch. These filters are defined by the user name, groups and possibly a list of site and area names. After authentication has been completed, these filters are used to limit the

search, or, in the process of site directory browsing (see below), to determine if a certain node is visible to this user, and whether the user can see node properties and/or change them.

Consider 2 cases:

### **1. Authentication is performed by Arch server (Solr) by the default reference Authenticator.**

First, the request source IP address is checked against the list of preconfigured blocked IP addresses, if any. If it matches any of them, access is blocked.

Then, if the user is not logged in yet and this is not a login request, the request source IP address is checked against the list of preconfigured allowed (backdoor) IP addresses, if any. If there is a match, the request is assigned static user, groups, site and area filters preconfigured for allowed IP addresses and access is granted.

If the IP address is not of the lists of blocked or allowed addresses, authentication is performed differently, depending on whether the "ar\_frontid" parameter is sent in the request.

If the "ar\_domain" parameter is sent in the request, the authentication configuration is taken from the configuration file of the site whose name matches the "ar\_domain" parameter. Otherwise, the authentication configuration is taken from the root configuration file. Note that if site specific authentication is used, all other sites are not visible to the client. This is because authentication is controlled by the site administrator in this case, not the root administrator.

**Case 1.a. No "ar\_frontid" parameter is sent in the request.** If there is no authentication information (user name and password) found in the request, the user is assigned the name "guest" and the group "public". If there is correct authentication information or the user has been authenticated before and this information is saved in the session object, the user is assigned their authenticated user name and groups. If the requested authentication information is incorrect (e.g. the password is wrong), authentication fails and access is blocked

**Case 1.b. "ar\_frontid" parameter is sent in the request.** Depending on the "ar\_domain" parameter or its absence, the Authenticator searches for a front-end profile record matching the value of "ar\_frontid" either in the root configuration file or in the configuration file of site defined by the "ar\_domain" value.

When the front-end is being authenticated, its profile is used to filter user names and group names sent with request. For example, a front-end operator may try to send user names or groups that have access to restricted information which is closed to users of that front-end. This attempt will be blocked by the server, unless these user and group names are listed as allowed in this front-end's profile. Front-ends can also be limited to search only particular areas or sites using their profiles.

**2. Authentication is performed by a PHP front-end.** In this case, the PHP front-end performs a similar authentication procedure to the described above, (except that the reference implementation is much simpler and does not support black and white lists), adds established user name and groups to the request and sends this request to Arch server, along with the front-end's ID and password.

### **Now the Arch server has to perform front-end authentication, as in case 1.b. 3.11 Site Directory**

An example of a site directory is presented in Figure 3.

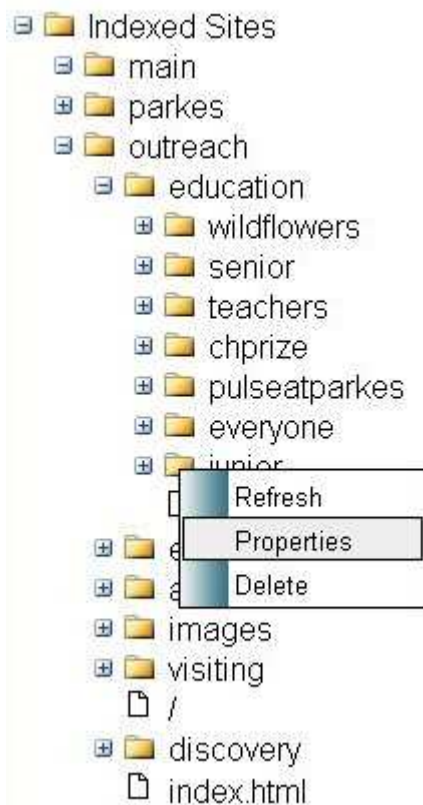


Figure 3. Arch site directory

Clicking on the small “+” image expands its folder by reading a level of nodes from the database via an AJAX request.

Right-clicking on folder or node image displays a context menu. Clicking on “Properties” in this menu reads node properties from the database and displays a node properties dialog as in Figure 4.

### Node Properties

**Name:** teachers

**Label:**

**Description:**

#### Permissions

Permissions inherited from parent folder

**Groups R/O:**

**Groups R/W:**

**Users R/O:**

**Users R/W:**

**Administrators:**

Figure 4. Arch node properties dialog

In this dialog:

**Name** is the real name of the document or folder.

**Label** is the text that is shown in the directory with the node. If label is empty, name is used.

**Description** is a short text that, if provided, is shown when user's mouse hovers over the node.

**Permissions inherited from parent folder** – if this box is checked, permissions settings are inherited from the parent folder.

**Users R/O** – a list of names of users who have read access to this node. They can see this node in the directory and find the document using a query. They can't see or change the node's properties. This is because user and groups names may be confidential.

**Users R/W** – a list of users who have read and write access to this node. In addition to the above R/O functions, they can see the node's properties and change label and description. They can't change access permissions. To change these, one has to be an administrator.

**Groups R/O** – user groups that have R/O access. See above.

**Groups R/W** – user groups that have R/W access. See above.

**Administrators** – users that have R/W access and can also change access permissions.

### 3.12 Internationalization

Arch was built with easy internationalization in mind. All text that is visible to users is localized in a handful of files. Language dependent files are selected based on the "lang" request parameter. If there is no such parameter, English is used. Depending on whether users see Solr or a PHP front-end, different files have to be translated.

1. To translate Solr output, copy Velocity template files from `$NUTCH_HOME/conf/arch/language/en/` directory to `$NUTCH_HOME/conf/arch/language/<other-language-code>/` directory and translate them.

2. To translate PHP, copy Smarty template files from `language/en/` directory to `language/<other-language-code>/` directory and translate them.

## 4 Evaluating Performance and Tuning Arch

Arch includes a tuning and evaluation module that allows assessing Arch performance against performance of other search engines using blind testing methodology and measure changes in performance produced by various modifications of algorithms and/or configuration parameters.

For evaluation, the module can be configured to submit user queries to several search engines in parallel and then present received results to the user who will mark correct hits by clicking on their titles in the results page. The results are composed of top 10 hits from each engine in randomly interleaved order, so that the user can't tell which engine returned which hits. After submitting several queries and marking correct hits for each query, the user clicks on the Finish button and performance figures are shown for each engine.

Test results			
Query	Google scores	Nutch1.1 scores	Arch scores
Mopra telescope observation seasons	10%	10%	20%
Parkes observatory open days	60%	50%	100%
Parkes open day hours	50%	30%	40%
atnf visitor accommodation	90%	40%	70%
use atnf telescopes by school students	30%	30%	50%
available computer clusters	0%	0%	0%
atnf available disk storage	10%	0%	0%
<b>Average precision over 7 queries</b>	<b>35.7%</b>	<b>22.9%</b>	<b>40%</b>

Figure 5. Example of test results page

For tuning, the queries and relevance judgements produced in the process of testing and saved to a file are used to show side-by-side results returned by the search engines in response to a given query. This process can be used to study effects of various changes on the count and distribution of correct hits.

Query: Calibrators, Scores shown: (rel.@10, rel.@50) (av. rel.@10, av. rel.@50)			
Arch (4, 5) (4, 4.67)	Nutch1.1 (2, 5) (2.33, 3.67)	Panoptic (1, 3) (1.78, 2.78)	Google (4, 4) (3.44, 4.44)
1. Calibrators	1. Calibrators	1. Note on Calibrators for the LBA	1. Calibrators
2. Calibrator cycle time calculator	2. Note on Calibrators for the LBA	2. Flux density calibrators	2. Note on Calibrators for the LBA
3. Passband/CACAL calibrators	3. Passband/CACAL calibrators	3. Long-term Monitoring of Molonglo Calibrators	3. Long-term Monitoring of Molonglo Calibrators
4. Characterisation of candidate calibrator	4. Calibrator cycle time calculator	4. Calibrators	4. Flux density calibrators
5. Calibrators	5. VLBI at the ATNF	5. VLBI at the ATNF	5. ATCA Calibrator database
6. ATCA Calibrator Catalogue	6. Calibrators	6. ATCA Calibrators	6. C007 project
7. ATCA Calibrator database	7. C007 project	7. CORRECTING THE FLUX DENSITY SCALE OF SECONDARY CALIBRATORS	7. Calibrators
8. Flux density calibrators	8. Characterisation of candidate calibrator	8. ATCA_calibrators	8. Passband/CACAL calibrators
9. ATCA calibrator catalogue update	9. ATCA calibrator catalogue update	9. ATCA Calibrators	9. ATCA Calibrators
10. ATCA Calibrators	10. 7mm Calibration Strategies	10. Long-term Monitoring of Molonglo Calibrators	10. ATCA calibrator catalogue update
11. Long-term Monitoring of Molonglo Calibrators	11. CORRECTING THE FLUX DENSITY SCALE OF SECONDARY CALIBRATORS	11. ATCA Calibrator database	11. CORRECTING THE FLUX DENSITY SCALE OF SECONDARY CALIBRATORS
12. C007 project	12. Astronomical Tools and Software	12. C007 project	12. ATCA Users Guide: 1.5 Millimetre-wave observations (12mm@3mm)
13. CORRECTING THE FLUX DENSITY SCALE OF SECONDARY CALIBRATORS	13. 3mm Calibration Strategies	13. Publications of the Astronomical Society of Australia	13. ATCA Users Guide: 1.4 Centimetre Observations (20@3 cm bands)
14. Note on Calibrators for the LBA	14. ATCA Calibrator Catalogue	14. SCHED	14. ATCA Users Guide: 4.3 Data Analysis
15. Long-term Monitoring of Molonglo Calibrators	15. ATCA Calibrators	15. ATNF Astronomy Web Services	15. ATCA Users Guide: 2.3 Scheduling Strategy
16. 7mm Calibration Strategies	16. 22 GHz observing sessions	16. Astronomical Tools and Software	16. ATCA_calibrators
17. 22 GHz observing sessions	17. 1934 dhi	17. ATCA calibrator catalogue update	17. Calibrators
18. 1934 dhi	18. http://www.atnf.csiro.au/vlbi/LBA-Novices-Manual/v1.4.pdf	18. Characterisation of candidate calibrator	18. SCHED
19. 3mm Calibration Strategies	19. userhtml	19. Calibrator cycle time calculator	19. MNRF Upgrade: 12-mm System on the ATCA
20. Calibration	20. SCHED	20. MNRF Phase Correction Project Details	20. Schedule - 26 Mar - 8 Apr Australia Telescope Compact Array
21. ATCA Users Guide: Index	21. Australia Telescope Compact Array Users Guide	21. Observations	21. Observations and Data Analysis
22. Calibration of Data with XX, YY, XY and YX	22. ATNF Compact Array Data Acquisition Problems and Information	22. ATNF Astronomy Web Services	22. Neri - A survey of radio sources at 86 GHz
23. An alternative approach to calibration	23. ATCA Calibrator database	23. Archive of VLBI Observations Majordomo List	23. mm Calibration Meeting 27 October 1997
24. Bandpass and antenna gain calibration	24. ATCA Current Issues	24. Introduction	24. Schedule - 26 Apr - 9 May Australia Telescope Compact Array
	25. DA Checklist	25. Results	
		26. Archive of VLBI Observations Majordomo List	

Figure 6. Example of tuning results page

The page on Figure 6 shows titles of hits returned in response to query “Calibrators” by Arch, Nutch 1.1, Panoptic (Funnelback) and Google. Correct hits are highlighted by red colour. For each engine, the page shows number of correct hits in top ten and fifty documents, and average numbers of correct hits in top ten and fifty documents over all queries submitted so far.

## CSIRO Astronomy and Space Science

Contact Arkadi Kosmyrin  
 Phone +61 2 9372 4633  
 Fax +61 2 9372 4444  
 Email Arkadi.Kosmyrin@csiro.au