Idler

Bryan Horling
Department of Computer Science
University of Massachusetts

DIS Lab Technical Memo 96-03

11 December, 1996

1 Description

Idler is a utility used to make incremental additions and refinements to a server characterization database, such as the once produced by ServDB. It does so by using the base knowledge (i.e. that located in the persistent database) to both increase the quality and accuracy of existent data and to help discover new, supposedly useful, information. The concept behind this program is that when the server database is actively being used (in a search), it is unlikely that the calling program would expend the extra resources necessary to fully analyze the information it retrieves[1]. For such a search to be efficient, though, the server database must be up-to-date, accurate, and as complete as possible. Idler (so named because it is expected to be used during idle time) solves this problem by addressing this need off-line[2], when more resources are available.

As idler runs, it generates a record similar to those stored in the server database for the particular site (URL) it is examining. It produces this record based on a physical retrieval of the information in question, which is then analyzed and characterized in several ways. Upon completion, the record is submitted to the database for storage.

Several types of data, separated into distinct record fields, are obtained during this process. Specific examples of such fields can be found in [2], and are are also

[1]It is obvious that some detailed analyzation will take place during the search, the results of which could be used to improve the database, but it is unclear that the scope of such analyzation will be able to provide the breadth and detail desired when updating the database

[2]An inactive time, not when we aren't connected to a network

listed in the Trait Calculation section. Some of this data is determined within Idler itself, but many fields are also determined through analyzation by auxiliary programs, or by the retrieval agent itself [1].

1.1 Statistical Data

The statistical data associated with a particular server characterizes the directly quantifiable traits which affect how the server interacts with a host. Such things include, but are not limited to, response latency, average monetary cost, information recency and data size. Given sufficient analysis, less tangible traits such as information quality can also be considered statistical evidence.

All of these items are computed through repeated queries to the server, with the expected value resulting from the average value encountered. The actual calculation of these values is left as a database function.

A problem inherent in this technique for gathering average data is that the "state of the world" will usually not be the same during information gathering as during an active search. Assuming that an active search will take place during a consistently similar time of day, and that an idle-time database modification will also happen at a consistent time of day different from that search time, we can see that the average values returned by such an idle-time search may not be indicative of how the server will react at the time when they are queried. Thus, it is imperative either to separate statistical data based on time, or to ensure that data is also gathered (at a similar rate) during the time in which an active search process is likely to occur. Highly variable or time-dependent data, such as server latency and retrieval cost, are most likely to be affected by this artifact.

1.2 Logical Data

The logical data examined from a server tries to distill the informational content which the server provides. In some sense, this may cover the information quality mentioned above, but its primary purpose is to describe what can be gleaned from the server, not necessarily how useful that information might be.

This data is gathered by inspection of the documents returned by the server. As each document is processed, the key terms[3] are extracted and stored in the server's record for later retrieval. The classification of the server (based on its document structure or server function) is also stored.

[3] The importance value implied by the phrase "key terms" is relative to the client, not the server.

2 Implementation

As described previously, Idler works by both updating and augmenting the data resident in the current server database. The process progresses by sequentially retrieving all of the records resident in the database, each time spawning a child process which performs an examination of an individual URL. The number of active child processes is bounded from above by the `MAX_CHILDREN` constant. Once a URL has been successfully retrieved and analyzed, one or more items may be added to or removed from the database, depending on the type and content of document retrieved.

Upon program commencement, a previously saved state is searched for. If found, its contained values are used to restore the state Idler was in when it was last run[4]. In addition, if such a previous state exists, the viewed URLs list is not refreshed. If a state does not exist, the viewed URLs list is refreshed and updating begins at the first record in the database.

If a `SIGHUP` signal is sent to the parent process, the idler will re-read its configuration file and restart the updating process, removing all traces of its previous working state.

2.1 Information Gathering

Documents (URLs) are always placed into one or more categories when they are added to the database (the type is stored in its `Category` field). As Idler runs, it looks and acts upon each URL as if it were a member of a specific category set, the classification of which affects how the record is updated and used.

The most prevalent category of URLs are those which are members of the `STATIC` category. By "static" it is meant that the documents supposedly contain real information that can be used (thus the user can remain "static" on the page and actually learn something[5]). During this static phase, each known static URL is retrieved, analyzed and (possibly) added to the database. To deduce the correct course of action, it is determined if the content of the document contains any known keywords, which are listed in the local file `keywords.txt`. If it does, then the document's record is stored, if not, the record is discarded. Furthermore, if the record is discarded, and it is known that the URL's record was originally retrieved from the database, then it is removed from the database as well. If the record is stored, then all URLs (links) contained on that page are marked with the current

[4]Such a saved state file is generated when Idler is terminated prematurely by a `SIGINT` or `SIGTERM` signal.

[5]Note that the amount of possible learning varies quite widely.

search level and placed in a queue[6]. The child process then goes on to iteratively retrieve and examine all URLs in the queue until the `MAX_STATIC_LEVEL` search level is reached, at which point the process terminates. As URLs are retrieved, they are also stored in a communal "visited" list, so that any child will not visit a particular URL which has already been visited in the current update phase. If the URL already exists in the database, though, its `REFERENCES` count will still be incremented.

The second category type is `INDEX`, which attempts to capture the notion of a document consisting primarily of a list of links (i.e. very little informative content). Since index pages are usually compilations of URLs pointing to useful static pages, it is thought that the maximum search level arising from such a page need not be large to find truly useful information sources; in fact, such pages are typically their immediate descendants. In addition, the large number of links usually found on such a page could also cause the Idler to spend an inordinate amount of time searching its document space. One can solve this problem by creating a new constant, `MAX_INDEX_LEVEL`, limiting the search level associated with these pages. Aside from this more constrained search space, these pages, and all descendants, are treated in the same manner as static pages.

The final set of records are not truly URLs at all. These records belong to the `SEARCH` category, and represent interactive search engines which can be used to find new, unrelated information sources. These records are the same as others, but with two exceptions. First, their `URL` field actually contains a user-defined alias, corresponding to a engine query characterization record as described in [1]. Because of this, search engines must be added to the database by hand. Second, they are not required to maintain an appropriate `KEYWORD` list to remain in the database, as static and index pages must. During the search phase, each resident search engine is queried with all known keywords. Their responses are then pruned of any self-serving links, and all remaining links are added to the database as static sources. At the end of each search, a server record is also created (with fields such as cost, time, etc) and submitted to the database. At this point in time, the newly found links are not examined further, and the child process will terminate when all possible searches to an engine have completed.

2.2 Trait Calculation

The following is a list of the document traits (field) which are calculated during the Idler updating process:

[6] By "search level" it is meant the logical level of the page as related to the root URL. So the root's descendants would be at level one; their descendants would be at level two, and so on.

`URL` The complete URL of the retrieved page (without named link #keywords) or server alias is returned. It is obtained either directly from the database, or by its appearance in an ancestor document.

`Time` The server latency, as returned by the retrieval agent.

`References` This will always be one, covering the reference used to get to the page. The true number of references is maintained by the database.

`Cost` The document cost, as returned by the retrieval agent.

`Size` The document size, as returned by the retrieval agent.

`Recency` The document recency, as returned by the retrieval agent.

`Category` Newly found documents are marked by default as `STATIC`, as it is the most common document type. If the ratio of combined link name and URL string lengths to the document's total content size is greater than `INDEX_LINK_LIMIT`, then it will be marked as `INDEX`.

`Keywords` The tab delimited list of known keywords found within the document. Keyword matching is case insensitive, and partial matches are allowed. If any keywords are returned by the retrieval agent these are also included.

`Descendants` The list of server-local descendant pages, as culled from the links the document contains. This feature is typically deactivated if a document-centric database is in use.

References

[1] Horling, Bryan. "Retriever Functionality", DIS Lab Tech. Memo 96-01 (1996).

[2] Horling, Bryan. "ServDB", DIS Lab Tech. Memo 96-02 (1996).