Similarity-aware Web Content Management and Document Pre-fetching

Jitian Xiao
Edith Cowan University

Michael Collins
Edith Cowan University

Proceedings of the Fourth International Conference on Machine Learning and Cybernetics, Guangzhou, 18-21 August 2005

SIMILARITY-AWARE WEB CONTENT MANAGEMENT AND DOCUMENT PRE-FETCHING

JI-TIAN XIAO, MICHAEL COLLINS

School of Computer and Information Science, Edith Cowan University, 2 Bradford Street, Mount Lawley, WA 6050, Australia
E-MAIL: {j.xiao, m.collins}@ecu.edu.au

Abstract:

Web caching is intended to reduce network traffic, server load and user-perceived retrieval latency. Web pre-fetching, which can be considered as "active" caching, builds on regular web caching, minimizing further a web user's access delay. To be effective, however, the pre-fetching techniques must be able to predict subsequent web access with minimum computational overheads. This paper presents a similarity-based mechanism to support similarity-aware web document pre-fetching between proxy caches and browsing clients. We first define a set of measures to assess similarities between web documents, and then propose a multi-cache architecture to cache web documents based on those similarities. A predictor is developed to support the similarity-aware document pre-fetching algorithm. Preliminary experiments have shown that our predictor offers superior performance when compared with some existing prediction algorithms.

Keywords:

Similarity; web caching; document pre-fetching

1. Introduction

Restrictions inherent in the differences of bandwidth between remote and local access to web content impose additional costs when accessing remotely hosted web resources [1]. Content caching, in its various forms, is seen as a set of techniques based upon historical analysis and/or projection, to alleviate the effects of server bottlenecks and the vagaries of network traffic volume, thereby reducing latency experienced by a server, user or by client programs. Traditional caching, at its basic level, locally stores recently requested pages so they do not have to be retrieved subsequently every time each is accessed. In brief, recently requested pages or files are held, or cached, on a local, or less remote, server in anticipation that they will be accessed again by clients. Such caching does much to reduce repeat network traffic. Clearly, though, newly requested documents will never be contained in such a cache. Pre-fetching is an active technique that attempts to guess those documents that are likely to be requested when a page leading to them is accessed – success of this technique is measured as a "hit-ratio". However, in such guessing, there is a need for an effective balance to be achieved between user comfort and computational overheads – the extremes are: too little effort applied, resulting in too many on-demand-fetches, while too much effort results in too many pre-fetches. The consequence of either is that of slower response to a user.

Previous work by Xiao [2] in developing pre-fetching predictions between caching proxies and browsing clients was based on measures of similarity between web users established that pre-fetching is capable of increasing the hit-ratio. Xiao's work further established that organisation of the cache affects opportunities for successful pre-fetching. In this position paper we describe a means of similarity based content management to improve the relative performance of pre-fetching techniques based upon document similarity detection.

Pre-fetch caching in the context of this study will be based upon similarity detection and involve four phases. Similarities will be sought from previously cached documents employing several concurrently applied, but differing, algorithms to detect equivalences of, e.g. broad-content or keywords, images and picture-titles and links contained within pages under scrutiny. Similarities between web-pages, having been detected, will then be ranked for candidature to be fetched in anticipation of a user's intentions. Following the ranking exercise, content settings may be realized for sub-caches and pre-fetching may then proceed.

The rest of the paper is organized as follows. Section 2 defines the similarity measures. In Section 3, we propose a similarity based web cache architecture. Section 4 presents the similarity-aware web document pre-fetching algorithm, and Section 5 concludes the paper.

2. Similarity measurement and detection

The exercise of measuring similarities among documents follows two main streams: one uses a single relationship between documents[1] or data objects while the other uses multiple relationships. Early research used a single relationship to measure the similarity of data objects. In the original *vector space model* (VSM) [3], "terms" (e.g. key words or stems) were used to characterize queries and documents, yielding a document-term relationship matrix to compute similarities among terms and documents by taking the inner product of the two corresponding row or column vectors. Dice, Jaccard and Cosine [4] used such document-term relationships to measure the similarity of documents for retrieval and clustering purposes. Deerwester and Dumais [5, 6] saw that a document might not be well represented by its contained keywords and developed a Latent Semantic Index (LSI). In this, they apply a *singular vector decomposition* (SVD) method to map the document-term matrix into some lower dimensional matrix where each dimension associates with a hidden "concept", where any similarity of text objects (documents and queries) is measured by relationships to those "concepts" rather than the keywords they contained.

With the advent of Word Wide Web, relationships with document objects, e.g. their hyperlink relationships, were used to derive similarity; a mechanism employed by both Dean [7] and Kleinberg [8] to discover similar web pages. Further, Larson [9] and Pitknow [10] applied co-citation to a hyperlink structure to measure any similarity of two web pages. Xiao [2] employed user-document access relationships to cluster users of similar interests. Flesca [11] proposed a method to measure the similarity of two documents that represents the current and the previous version of monitored pages for effective web change detection.

The approaches introduced above all relied upon a single relationship to measure any similarity of data objects. However, such approaches may run into serious problems when applications require accurate similarity e.g. where multiple types of data objects and relationships must be handled in an integrated manner. Accordingly, in the extended VSM [12], feature vectors of data objects were augmented by adding attributes from objects of other related spaces. Similarity computation is then obtained from calculation on these enhanced feature vectors. The extended feature vectors were used for document search [13] or clustering purposes [14]. Racchio [15] and Ide [16] expanded the query vector using those frequently-used terms appearing in the foremost documents retrieved by a query to improve search effectiveness. Similarly, Brauen [17] modified document vectors by related query terms.

Recently, it has been tried to calculate the similarity of two data objects based upon any similarity of their related data objects. Raghavan and Sever [18] tried to measure the similarity of two queries by correspondences found in their respective search lists. Beeferman and Berger [19] clustered queries using the similarity of both their selected web pages and cluster web pages based upon similarities of the queries that lead to the selection of those web pages. Both [20] and [21] calculated the query similarity based on both the query contents similarity and the similarity of the documents that were retrieved by the queries.

In this paper we define similarity measures of web documents for effective web document caching and pre-fetching. To pre-fetch documents that are of similar topic to the document a user is currently viewing, we need to derive the similarity of contents of web documents, ignoring any structural elements, e.g. HTML formatting. For efficacy of on-line pre-fetching, we propose different levels of similarity measures to capture levels of similarity between web documents. Consider a search of scientific papers over the web. A keyword based search usually returns a list of documents containing some or all of the given keywords. The matched keywords in the returned documents may appear in the title, keywords section, or other parts. Title/author-based searches follow similar principles. However, when a user is viewing a document and wishes to search for documents of similar topic, then the matching strategy may be quite different because the words to be matched may be related rather than explicitly stated. In our study, similarities between text documents are measured based on topics, page titles, keywords or page contents or combinations thereof. Compared with a keyword-based similarity measure, a content-based similarity is complicated by the need for special techniques, e.g., from the area of information retrieval [22]. However, any computation of similarity still needs to be completed within a reasonable time limit.

2.1. Document Model

To calculate similarities among web documents, we use a model based on the document model representation in [11], wherein structured web documents are represented as unordered labeled trees. That is, we consider containment rather than order of appearance of words within a document. However, our model differs from that in [11] in two ways: first, we don't consider the HTML formatting elements and,

[1] In this paper, a *document* refers to a text document or a web page that may contain text, images and/or pictures.

second, we consider a document's structure to be based on sectional elements, e.g. *Abstract* and *subsections*, while their work specifies texts in terms of pairs of start and end tags, e.g., <table> … </table>, <ul. … </ul>.

In the resultant tree, each non-leaf node corresponds to a subsection of the document (e.g. characterizing the title of the subsection), except that the root-node might also contain a set of *keywords*, a list of *authors*, a string for *title*, or/and a set of words comprising the *abstract*. Each leaf node corresponds to the text of that (sub)section. Notably, such a structure allows us to determine sectional similarities between particular elements such as titles; between the various contents, and, implicitly, between the structures of compared documents. In brief, then, a document tree is an unordered tree wherein each node is characterized by an associated set of type-value pairs.

Given a document tree *T*, of root *r*, with a node n_r we may represent a sub-tree of *T* rooted at n_r as $T(n_r)$. We define a set of functions, each characterizing some element, on the document tree: *keyword*(*r*), *title*(*r*), *authors*(*r*), *abstract*(*r*) and *text*(*r*). For a document tree rooted at *r*, *keyword*(*r*) = {*s* | *s* is a keyword contained in the keyword section of *r*}. The *title*(*r*), *authors*(*r*) and *abstract*(*r*) can be defined similarly. If $n_1, n_2, \ldots, n_k$ are child nodes of *r,* then

$$text(r)=\begin{cases} title(r) \cup \cup_{i=1}^{k}\{s \mid s \in text(n_i)\} & \text{if } r \text{ is a non-leaf node, with children } n_1, \ldots, n_k \\ \{s \mid s \text{ is a word in } leaf(T(r))\} & \text{if } r \text{ is a leaf node of } T \end{cases}$$

Essentially *text*(*r*) is a set of words contained in the various strings associated with nodes of the (sub-)tree rooted at *r*. Note that text(*r*) is defined recursively.

Our similarity calculation algorithm works on this tree structure by exploiting the information contained in individual nodes and the whole tree. Observe that each node keeps track of its level in the tree, its content and the content of its child nodes.

2.2. Levels of Document Similarity Measures

Levels of document similarity measures are defined by making use of the text extracted from elements of document (sub-)trees. To compute the similarities efficiently, the measures must be normalized, allowing the comparison of pairs of documents and the selection of different levels of elements/components. Given two document trees T_1 and T_2, and two nodes $r_1 \in T_1$ and $r_2 \in T_2$, we define

$$intersect\,(w(r_1), w(r_2)) = \frac{|\,w(r_1) \cap w(r_2)\,|}{|\,w(r_1) \cup w(r_2)\,|} \quad (1)$$

where *w(r)* is a set of strings associated with nodes of the (sub-)tree rooted at *r*. The function $intersect(w(r_1),w(r_2))$ returns the percentage of the number of common words divided by the number of all words that appear in both $w(r_1)$ and $w(r_2)$. Clearly, $intersect(w(r_1),w(r_2)) \le 1$, while equality exists when $w(r_1) = w(r_2)$.

For two document trees rooted at r_1 and r_2, respectively, similarities of keyword, title and abstract may be defined by the formulae (2) through (4):

$$SIM_{KB}(r_1, r_2) = intersect(keyword(r_1), keyword(r_2)) \quad (2)$$

$$SIM_{TB}(r_1, r_2) = intersect(title(r_1), title(r_2)) \quad (3)$$

$$SIM_{AB}(r_1, r_2) = intersect(abstract(r_1), abstract(r_2)) \quad (4)$$

while the content-based similarity is defined as

$$SIM_{CB}(r_1, r_2) = intersect(w(r_1), w(r_2)) \quad (5)$$

Where $w(r_i) = text(r_i) \cup keywords(r_i) \cup abstract(r_i)$, $1 \le i \le 2$.

Generally, the higher a word occurrence in a document, the closer that word relates to the theme of the document and this may be used as a measure of similarity. Let $weight_r(s)$ be the number of appearances of the word *s* in document represented by *r*, then the intersect function can be defined as

$$intersect_{wt}\,(w(r_1), w(r_2)) = \frac{\sum_{s \in w(r_1) \cap w(r_2)} \min\{weight_{r_1}(s), weight_{r_2}(s)\}}{\frac{1}{2}\sum_{s \in w(r_1) \cup w(r_2)} |\,weight_{r_1}(s) + weight_{r_2}(s)\,|} \quad (6)$$

Based on this function, the *weighted* similarity measures $SIM_{KB}(\,)$, $SIM_{TB}(\,)$, $SIM_{AB}(\,)$ and $SIM_{CB}(\,)$ can all be defined by replacing *intersect*( ) with $intersect_{wt}(\,)$ defined in (2) to (5) above.

2.3. Data pre-processing

To calculate similarities among documents, a text filter was developed to extract meaningful words from related sections of a document, and count them per section. The method is described briefly below:

In the text filter, raw text is first parsed into generalized words, called *tokens*. Tokens include meaningful strings, abbreviations, punctuation and other specialized symbols that have been derived from the structure found in the document's sections. For example, while typical words such as "web" and "page" are taken as tokens, the punctuation mark "$" and the URL "www.ecu.edu.au" are also tokens. However, digits and others insignificant words, e.g. pronouns and prepositions, are not treated as tokens.

For each section, the text filter produces a list of (*token, c(token)*) pairs, where *c(token)* is the count of that token within the section – in effect, a *bag-of-words* basis for our representation. Note that for brevity of the token list and subsequent comparison, each word is reduced to its stem (e.g., *server* and *service* into *serve*). While the unordered *bag-of-words* model will not suffice for linguistic analysis, we assume it captures most of the information needed for calculating similarities using formula (2) ~ (5).

3. Similarity-aware Web Content Management

The basic idea of web-caching is to reduce network traffic load and reduce retrieval latency by holding recent requested documents at the proxy caches so that they do not have to be fully retrieved upon identical request.

Document similarity information is fundamental to effective caching and pre-fetching, yet it has never been incorporated directly in cache replacement algorithms. Rather, other properties of the request stream (e.g., document size and access frequency etc.), being easier to capture on-line, are used to infer similarity, and hence driven cache replacement policies. In this section, we propose a similarity-based multi-cache web content management scheme and on-line algorithm to capture and maintain an apposite similarity profile of documents requested through a caching proxy and describe a novel cache replacement policy using such information to support the similarity-aware pre-fetching.

3.1. The caching architecture

We now present a similarity-based multi-cache web content management scheme. There are four major components: *central router*, *similarity profiles, sub-caches (SP),* and *document allocator*. Of these, the central router is pivotal in controlling and coordinating the other components.

Before configuring the multi-cache web content management scheme, we first cluster documents in cache based on the similarity measures introduced in (2)~(6), and determine the number of themes, N, of the documents. For each theme/cluster, a number of *stems* relating to it were chosen (e.g., by looking at all stems produced by the text filter when SP vectors were computed). Then the cache is divided into $N+1$ sub-caches. Each of the first N sub-caches stores documents of one particular theme, and the last sub-cache stores other documents not belonging to any of the N themes. In this way, we ensure that similarities among documents in any sub-cache are relatively higher, while relegating those among documents across sub-caches.

3.1.1. Similarity Profiles

The SP comprises N two-dimensional arrays $A_i(*, *)$, $i=1, 2, ..., N$, of which each corresponds to one of the first N sub-caches. For each document j in sub-cache i, SP counts the number of occurrences of the stems that relate to the theme of the sub-cache, storing the numbers in vector $A_i(j,*)$. This information is useful when performing similarity-aware pre-fetching from the sub-cache to a client. For each theme, we limit the number of stems to be 100.

3.1.2. Sub-caches

A sub-cache is an independent cache that has its own cache space, contents and replacement policy. Since documents in a same sub-cache are usually of similar theme, simpler replacement policies, e.g. LRU, LFU and FIFO, may be applied.

3.1.3. Sub-cache document allocator

The sub-cache allocator assesses comprehensively a candidate set of evictions selected by sub-caches, with possible results of: re-caching, eviction or probation. Of these, re-caching and eviction are instantaneous, while a probation document will be held by the allocator in its own space pending a final decision.

3.2. Algorithm Framework of Similarity-aware Content Management (SACM)

A request for a document d invokes the SACM algorithm as follows: an instance of d is sought in an in-cache index; if d is already cached (cache hit) and still fresh its containing sub-cache is noted whereupon d will be returned to the requesting client. If the instance of d is not fresh, then re-cache from an origin server, updating related parameters such as SP vectors. For a cache miss, the request for d will be forwarded to the origin server and a resultant downloaded document d_{new}, is returned to the client. Based on the content of d_{new}, a SP vector will be calculated to determine a sub-cache i_d in which d_{new} is to be cached. Where there is insufficient space for d_{new}, then sub cache i_d makes room according to its eviction (e.g. LRU, LFU) and/or space sharing policies. The document allocator of i_d will then assess and purge any eviction candidates.

The central router mediates between cooperating sub-caches. Although a document may be cached "conceptually" in several sub-caches in terms of sub-cache document allocator evaluation, only one actual copy will be maintained.

4. Similarity-aware document pre-fetching

In this section, we focus on any pre-fetching between caching proxies and browsing clients in idle periods of their network links when a current web document is read by a user. If the proxy can predict those cached documents a user might access, the idle periods may be used to push them to the user, or to have the browser/client pull them. Since the proxy only initiates pre-fetches for documents in its caches, there is no extra internet traffic increase.

We propose two similarity-based algorithms to guide pre-fetching from proxy caches to clients. The first one is a pure similarity-based pre-fetcher which considers only those documents whose similarities with the current viewing document surpass a certain threshold. The second algorithm (i.e., similarity-aware pre-fetching) combines the *prediction by partial matching (PPM) method* [1] and the pure similarity-based pre-fetching strategies.

4.1. Similarity-based pre-fetching predictor

The similarity-based pre-fetching predictor evaluates the next k documents in the cache based on document similarities.

With the support of the similarity-aware web cache architecture, our similarity-based document pre-fetching predictor works based on a very simple rule. Suppose a client is viewing a document, say p (at this time, a copy of p must be cached in a certain sub-cache, say i, or being held by the allocator). Then the pre-fetching predictor will calculate the similarities between p and those documents in sub-cache i by referencing the similarity information in i_{th} SP. No documents in other sub-caches are considered because of their low similarities with p. Then the predictor simply chooses k documents whose similarities with p are among the top k highest ones. These k documents, together with those cached pages to which hyperlinks exist from p, will be returned to the *pre-fetcher* for the possibility of pre-fetching.

4.2. Similarity-aware pre-fetching predictor

The PPM developed in [1] essentially predicts the next l requests on the past m accesses of a user, limiting candidates by an access probability threshold t. The performance metrics of the algorithm depend on the (m, l, t) configurations. However, the algorithm uses patterns observed from all users' references to predict a particular user's behavior. Referencing too many contexts makes the prediction inaccurate, inefficient and unwieldy.

Our previous work [2] extended the PPM algorithm by referencing only those access patterns from a small group of other users exhibiting high similarities in their past access patterns to predict a current user's next access. The number of times the algorithm can make prediction is reduced because of the smaller sample size, but the hit ratio of the pre-fetching increases because more related access patterns are referenced. We call the method *pattern-similarity based PPM* (or *psPPM*).

To be more similarity-aware, we now modify *PPM* and *psPPM* by replacing the access threshold t with s, where s is the similarity threshold between the document to be pre-fetched and the document the client is viewing. Thus the new algorithm has the following parameters:

- r: the number of users whose access patterns are referenced to predict future accesses of the current user.
- m: the number of past accesses that are used to predict future ones. We call m the *prefix depth*.
- l: the number of steps that the algorithm tries to predict into the future.
- s: the similarity threshold used to weed out candidate document. Only those documents whose similarity with the viewing document is greater than s, where $0 \leq s \leq 1$, is considered for pre-fetching.

Suppose a user u is viewing a document d. A set of r users whose access patterns showed relatively high similarities with u is evaluated and ordered in descending order. For l>1, not only the immediate next request, but the next few requests after an URL are also considered for potential pre-fetching. For example, if $l=2$, the algorithm predicts both the immediate next and its successor for the user. If m>1, more contexts of the r users' past accesses are referenced for the purpose of improving the accuracy of the prediction.

The predictor maintains a data structure that tracks the sequence of URLs for every user. For prediction, the past reference, the past two references, and up to the past m references are matched against the collection of succession to the users' past access patterns to produce a list of URLs for the next l steps. If a longer match sequence can be found from the other r users' patterns, the next URL to the longest match is also taken as a potential document to be accessed next by the user. The outcome of each prediction is a list of candidate documents, ordered by their similarities with d. For those candidate documents with the same similarity value, the URL matched with longer prefix is put first in the list.

We conducted two series of preliminary simulations. The first series of simulations is to demonstrate the capability of our similarity measures for document comparison to determine the document themes (or clusters). Using the obtained similarity information, our second series of simulations demonstrates the improvement in prediction accuracy (and thus hit rate) of the pre-fetching between caching proxies and browsing users using our similarity-based/aware predictors. The preliminary results indicate that our predictor is capable of practical prediction for web document pre-fetching in the sense and an improvement of the order of 10% over traditional PPM. We intend to perform more extensive simulations on real Web log data, of which the results will be published in future.

5. Conclusions

We proposed a similarity-aware web content management scheme, presented its underlying algorithms and developed a similarity-aware predictor for web document pre-fetching between proxy caches and browsing clients. Simulations indicate that our predictor is capable of practical prediction for web document pre-fetching in the sense that it may predict more accurately and rapidly than the traditional PPM does by only referencing to a reduced set of users' past access patterns.

References

[1] L. Fan, P. Cao, W. Lin and Q. Jacobson, Web Prefetching between Low-Bandwidth Client and Proxies: Potential and Performance, *SIGMETRICS'99*, 1999.

[2] Xiao, J., Zhang, Y., Jia, X., and T. Li. Measuring Similarity of Interests for Clustering Web-Users. *Proceedings of the* 12th *Australian Database Conference 2001 (ADC'2001).* Gold Coast, Australia, 107-114, 2001.

[3] Salton. G., *Automatic Information Organization and Retrieval*. McGraw-Hill, 1968.

[4] Rasmussen, E., Clustering algorithms. Information Retrieval: Data Structure and Algorithms. Prentice Hall, 419-442, 1992.

[5] Deerwester, S., Dumais, S.T., Landauer, T.K., Furnas, G.W., and Harshman., R.A., Indexing by Latent Semantics Analysis, Journal of the Society for Information Science, 41(6), 391-407.

[6] Dumais, S.T., Furnas, G.W., Landauer, T.K., and Deerwester, S., Using Latent Semantic Analysis to Improve Information Retrieval, *Proceedings of the CHI'88: Conference on Human Factors in Computing Systems,* New York, ACM, 281-285.

[7] Dean, J., and Henzinger, M.R., finding Related Pages in the World-Wide Web. Proceedings of the 8th International Conference on World Wide Web, 1999.

[8] Kleinberg, J.M., *Authoritative Sources in a Hyperlinked Environment*, J. of the ACM (JACM), 46(5). 604-632.

[9] Larson, R.R., Bibliometrics of the World-Wide Web: An Exploratory Analysis of the Intellectual Structure of Cyberspace. *Proceedings of the Annual Meeting of the American Society for Information Science,* Baltimore, Maryland, 1996.

[10] Pitkow, J. and Pirolli, P., Life, Death, and Lawfulness on the Electronic Frontier. *Proceedings of the Conference on Human Factors in Computing Systems,* Atlanda, Georgia, 1997.

[11] Flesca, S. and Masciari, E. Efficient and Effective Web Change Detection, *Data & Knowledge Engineering*, Elsevier, 2003.

[12] Fox, E., Extending the Boolean and Vector Space Models on Information Retrieval with P-Norm Queries and Multiple Concepts Types. Cornell University Dissertation.

[13] Shaw, J.A., and Fox E.A., Combination of Multiple Searches. *Proceedings of the 3rd Text Retrieval Conference (TREC-3),* 1994, 105.

[14] Chakrabarti, S., Dom, B.E., Kumar, S.R, Raghavan, P., Rajagopalan, S., Tomkins, A., Gibson, D. and Kleinberg, J.M., Mining the Web's Link Structure, *IEEE Computer*, 32 (8). 60-67.

[15] Rocchio, J.J. and McGill, M.J., *Relevance Feedback in Information Retrieval.* Prentice-Hall Inc., Englewood Cliff, NJ, 1997.

[16] Ide, E., *New Experiments in Relevance Feedback*, Prentice-Hall, 1971.

[17] Brauen, T., *Document Vector Modification*, Prentice-Hall Inc., Englewood Cliff, New Jersey, 1971.

[18] Popescul, A., Flake, G., Lawrence, S., Ungar, L.H., and Gile, C.L., Clustering and Identifying Temporal Trends in Document Database. *Proceedings of the IEEE advances in Digital Libraries*, Washington,2000.

[19] Beefermand, D., Berger, A., Agglomerative clustering of a search engine query log. Proceedings of the sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Boston, MA, 407-415, 2000.

[20] Wen, J.R., Nie, J.Y., and Zhang, H.J., Querying Clustering Using User Logs., *ACM Transactions on Information Systems* (TOIS), 20(1), 59-81, 2002.

[21] Su, Z., Yang, Q, Zhang, H.J., Xu, X., and Hu, Y., Correlation–Based Document Clustering Using Web Logs. Proceedings of the 34th Hawaii International Conference on System Science, Hawaii, USA, 2001.

[22] Su, Z., Yang, Q, Zhang, H.J., Xu, X., and Hu, Y., Correlation–Based Document Clustering Using Web Logs. Proceedings of the 34th Hawaii International Conference on System Science, Hawaii, USA, 2001.