

# Evaluating Communication Strategies in a Multi Agent Information Retrieval System

David Lillis<sup>1</sup>, Rem Collier<sup>1</sup>, Fergus Toolan<sup>2</sup>, and John Dunnion<sup>1</sup>

<sup>1</sup> School of Computer Science and Informatics  
University College Dublin

{david.lillis, rem.collier, john.dunnion}@ucd.ie

<sup>2</sup> Faculty of Computing Science  
Griffith College Dublin  
fergus.toolan@gcd.ie

**Abstract.** With the complexity of computer systems increasing with time, the need for systems that are capable of managing themselves has become an important consideration in the Information Technology industry. In this paper, we discuss HOTAIR: a scalable, autonomic Multi-Agent Information Retrieval System. In particular, we focus on the incorporation of self-configuring and self-optimising features into the system. We investigate two alternative methods by which the system can configure itself in order to perform its task. We also discuss the Performance Management element, whose aim is to optimise system performance.

## 1 Introduction

Complexity is a serious issue with modern computer systems. As hardware has improved dramatically to facilitate the development of more and more powerful systems, the complexity of the software that runs on it has increased accordingly. As a result of this, the costs associated with administrating such systems has become a serious issue in the Information Technology industry.

This has resulted in a push towards the development of software systems that are capable of managing themselves, in order to reduce the input required from human administrators. Research in this area has come to be referred to as Autonomic Computing, a term coined by IBM in 2001 [1]. Essential features of an autonomic computing system include self-configuration, self-optimisation, self-protection, self-healing and a number of others [2–4].

In addition to these hardware and software changes, another feature of the computer industry in recent years has been the widespread adoption of the World Wide Web. This has resulted in a dramatic increase in the quantity of information being made available, through such things as news articles, blog entries and forum postings. As a result, Information Retrieval (IR) systems must be capable of dealing with more and more information, which has resulted the need for more sophisticated systems. Efficiency and scalability have been of increasing concern in the IR community, along with the presentation of high-quality results.

The development of IR systems using multi-agent techniques is not a new phenomenon [5–7]. Indeed, the aim of our work on HOTAIR (Highly Organised Team of Agents for Information Retrieval), a multi agent IR system with autonomic features [8], is not to demonstrate innovation in the development of multi-agent IR systems, but rather to investigate a range of techniques that will enhance the robustness and scalability of large-scale agent systems. Here, we focus on two approaches to agent interaction that facilitate self-configuration.

Accordingly, Section 2 outlines reasons why IR is an ideal testbed for an Autonomic Computing System. Section 3 presents a general overview of the HOTAIR architecture, of which two versions have been developed. The first incorporates a Broker agent which maintains a centralised view of the entire system and manages the behaviour of other agents. An alternative architecture allows individual agents more autonomy by allowing them to gather more knowledge about their environment. Each version features a Performance Manager to provide elements of self-optimisation. Section 4 outlines experiments carried out to compare the performance of these two architectures. Finally, we include our conclusions and ideas for future work in Section 5.

## 2 Information Retrieval as an Autonomic Computing Testbed

An IR system is an ideal testbed for autonomic computing for a number of reasons. Firstly, a number of essential features of autonomic computing systems also have great relevance to IR systems:

- **Self-Configuration:** As more and more information becomes available to be processed by IR systems, it becomes necessary for the systems themselves to grow accordingly. This necessitates the introduction of additional hardware into the system, to cope with the increased workload. As it would be unfeasible to merely transfer the data from one system to another, it is necessary to add this hardware while the system is running. A self-configuring system would be capable of incorporating these additional resources into the system and make use of them, without intervention by a human administrator.
- **Self-Optimisation:** As with any large-scale system, optimal use of the available resources is an important aim. A self-optimising system will monitor its own use of resources and can react immediately to exploit any opportunities for greater performance that may arise.
- **Self-Healing and Self-Protection:** Recovery from failures and protection from external attack are essential to ensure reliability in any large-scale system, including an IR system. Given the huge revenues that are earned in advertising alongside, for example, online searches, prolonged downtime could have disastrous consequences for popular IR systems.

In addition to the above, the IR domain is an attractive one in which to carry out research on autonomic computing. The barriers to entry are low, as an IR system can be run on a cluster of low-end desktop computers. Also, the

existence of open source IR libraries such as Lucene<sup>3</sup> and Xapian<sup>4</sup> mean that the learning curve in setting up an IR system is relatively shallow in comparison to some other domains.

### 3 The HOTAIR Architecture

HOTAIR is an Information Retrieval system developed as a Multi-Agent System. It is written in the AFAPL2 agent programming language [9] and runs within the Agent Factory framework, a FIPA-compliant runtime environment for agents [10, 11]. The aim of the HOTAIR project is the development of a reliable and scalable agent-based search engine architecture. As discussed in Section 2, this application domain was chosen because we believe that IR is a problem that offers significant challenges in terms of the scale of the applications, which invites the use of autonomic features. Consistent with a typical IR system, HOTAIR consists of two distinct subsystems:

- **Indexing Subsystem:** This is responsible for identifying new documents to make available to users, whether from the World Wide Web, an FTP site, a ZIP archive, a file share or some other source. These documents must be added to a searchable index from which results will be extracted to be presented to users.
- **Querying Subsystem:** This is responsible for accepting queries from users, running those queries against the index built up by the Indexing Subsystem and returning those results to the user.

The focus of this paper is on the Indexing Subsystem. In order for a document to be added to the index, it must go through three stages. Each of these three steps is carried out by what is referred to as a “core agent”. *DataGatherer* agents are responsible for identifying and collecting new documents for inclusion in the index. These documents may be taken from any number of sources and may be in a variety of file formats. *Translators* are necessary to interpret the documents of various file types that have been collected by the *DataGatherers*. These are converted into a common file type, known as the HOTAIR Document Format (HDF), which maintains an XML representation of the document contents. *Indexers* represent the final step that a document must go through in order to be included in the system’s index. Indexer agents accept HDF documents as their inputs, extract the contents from these documents and save this data in the index. In order to reduce the amount of communication necessary between agents, documents are not processed individually, but rather in bundles of 20. We refer to these bundles as “jobs”.

Initially, core agents are not aware of the location of other core agents. The discovery of, and interaction with, other relevant core agents is an element of self-configuration which we aim to introduce in the following sections. During

---

<sup>3</sup> <http://lucene.apache.org>

<sup>4</sup> <http://www.xapian.org>

the development process, two versions of the HOTAIR architecture were created. The principal difference between these is the way in which agents gain knowledge about their environment and discover other agents with which they must interact in order to carry out their tasks. Section 3.1 describes the basic workflow of the system, which both architectures share. Of the two architectures, the “broker architecture” uses a Broker agent to micro-manage the behaviour of the core agents. This is presented in Section 3.2. Section 3.3 presents the alternative “broadcast architecture”, in which the core agents have more control over their own behaviour, as they have access to more information about the state of other agents and of the system as a whole. Finally, section 3.4 describes the performance management features that are applicable to both architectures.

### 3.1 Indexing System Workflow

This section describes the workflow of the HOTAIR Indexing Subsystem: how the core agents interact with one another in order to include documents in the index. Activities discussed in this section are carried out by core agents in both the brokered and broadcast architectures. We use the term “Processor” to describe any agent that receives jobs from another agent and processes them in some way (i.e. Translators process jobs taken from DataGatherers and Indexers process jobs taken from Translators). “Provider” is a generic term to refer to any agent that provides jobs for a Processor (i.e. DataGatherers provide jobs for Translators, which in turn provide jobs for Indexers).

At any given time, each Processor is assigned to a single Provider, from which it receives documents for processing. The way in which this assignment takes place is the fundamental difference between the brokered and broadcast architectures and is discussed in detail in the following sections.

The actual flow of jobs through the system then follows a pull-style pattern. When a Processor has the capacity to process a job, it requests a job from the Provider to which it is assigned. Each Provider maintains an output queue, which contains jobs on which it has completed its own processing. It is from this queue that a job is taken and forwarded to the Processor that requested it. If the Provider’s output queue is empty, it will reply with that information, at which time the Processor must be reassigned to an alternative Provider. The Performance Manager (discussed in Section 3.4) is charged with ensuring that the system is balanced so there are always documents available for processing.

Once the Processor has finished processing the job, it adds the job to its output queue if it is also a Provider. Indexers do not have output queues, as they represent the final stage of processing a document must undergo.

### 3.2 Brokered Architecture

The Brokered version of HOTAIR employs a Broker agent to aid the core agents in the discovery of the other core agents with which they need to communicate

and interact. Brokers have long been seen as a useful design pattern for creating more open agent systems [12, 13]. Here, the core agents have no first-hand knowledge of their environment other than the location of the Broker agent.

Upon creation, each new core agent must register with the Broker and is included in the model the Broker uses to assign Processors to Providers. Each Provider periodically contacts the Broker to inform it of the number of jobs currently contained in its output queue. This allows the Broker to build a model of the amount of work being created for each category of Processor.

In order to find jobs to process, a Processor must contact the Broker and request that it be assigned to a Provider. The Broker then makes use of its knowledge of the output queue status of the relevant Providers and of the assignments it has already made to make the most appropriate assignment. A Processor will continue requesting jobs from the same Provider until either the Broker reassigns it to a different Provider or the Provider informs it that its output queue is empty. In the latter case, the Processor must re-contact the Broker to be reassigned to an alternative Provider.

A key advantage of such an architecture is that all agent assignments are made with full knowledge of the state of the system as a whole. Thus, the allocation of agents can at all times be balanced so as to ensure that all Providers will eventually have Processors assigned to them. Assuming the Broker is using an appropriate assignment algorithm, this has the effect that jobs cannot become held up in a situation where they are located in the output queue of a Provider that never has a Processor assigned to it.

This form of Broker agent does, however, introduce a single point of failure into the system. If Broker fails or becomes uncontactable, the system as a whole will also fail. Core agents will no longer be able to identify Providers, as they will not have a method of finding their location. As the Broker is also the sole agent that is aware of the type, location and status of the core agents, this information is also lost if the Broker fails. In order to overcome this limitation, work has been carried out on the development of robust brokered architectures in order to maintain system performance in the event of the failure of the Broker [14].

### 3.3 Broadcast Architecture

The second version of the HOTAIR architecture involved Providers broadcasting the state of their output queues to all other core agents, rather than just to a centralised Broker agent. Specifically, a wild card (“\*”) was introduced into the agent name component of the FIPA agent identifiers allowing the partial specification of receiver agents. An example of this would be the agent identifier *agentID(ind\*, addresses(http://localhost:444/acc))*<sup>5</sup>, which could be used to send a message to all agents on the specified agent platform whose name begins with “ind” (which, in conjunction with an appropriate naming scheme, could be used to contact all Indexer agents). Furthermore, replacing “ind\*” with “\*”

---

<sup>5</sup> This identifier is specified in the format that is employed by the AFAPL programming language

is akin to a broadcast to all agents on the specified agent platform. Following this, a UDP multicast MTS was introduced, which, together with the wild card, allowed broadcasting of message to all agents on all platforms that are listening via the relevant UDP agent communication channel.

By introducing the above message broadcasting mechanism, and allowing Providers to broadcast their state, the need for the Broker agent is removed. Each core agent can build its own model of its environment and decide for itself which Provider from which it will request jobs. Perhaps the key issue in employing a UDP-based agent communication channel is that agent communication is no longer guaranteed. However, so long as the approach is used judiciously (e.g. repeated broadcasting of status updates) and in conjunction with guaranteed communication mechanisms (e.g. for job requests) the overall behaviour of the system can be guaranteed without affecting the robustness of the system. For some tasks, however, it would be useful to have the ability to broadcast reliably, and so we intend to investigate alternative methods of broadcasting using XMPP-based technologies such as Jabber<sup>6</sup>.

The broadcasts made by core agents include information about the length of their output queue, the number of documents in jobs processed and the time taken to perform this processing. In addition to being used by other core agents to support the system workflow, it is also possible for a Performance Manager to make use of this information to influence its management decisions.

Using this broadcast architecture, assignment of Processors to Providers is carried out by the Processors themselves. By default, agents assign themselves to the Provider that has most recently broadcasted the largest output queue. This self-assignment allows the agents to maintain the functioning of the system even in the absence of any management agents, thus removing the single point of failure without the need to introduce additional strategies to improve the robustness of the Broker. This self-assignment can, however, be overridden by instructions from a Performance Manager agent to assign it to another Provider.

### 3.4 Performance Management

Self-optimisation of the system is carried out by a Performance Manager. The aim of this manager is to maximise throughput. In order to carry out this task, it has a number of actions available to it so as to alter system behaviour.

**Agent Reassignment:** In the broadcast architecture, agents will by default assign themselves to Providers by selecting the Provider with the greatest number of outstanding jobs. Once this self-assignment has taken place, the Processor informs the Performance Manager of its decision, so as to ensure that the Performance Manager is fully aware of the current system configuration.

While such behaviour ensures that the system continues functioning, it does not necessarily lead to optimal system performance. A given agent only has knowledge of the Providers available to it, and is unaware of the behaviour of its peers. Thus the behaviour of an individual agent is not optimised for the system

---

<sup>6</sup> <http://www.jabber.org>

as a whole. As an example, if multiple agents seek a Provider simultaneously, each will assign itself to the same Provider (i.e. that which has most recently advertised the greatest number of available jobs). In some situations, the number of agents contacting the Provider is greater than the number of jobs the Provider has available. When this occurs, some agents must assign themselves to another Provider without having processed any jobs sourced from the initial Provider.

To counteract this, the Performance Manager can override the default behaviour by instructing agents to reassign themselves to other Providers. The decision to reassign Processors is currently based on the distribution of outstanding jobs in each Provider group's output queues. When deciding which agents to assign to a particular Provider, the Performance Manager prioritises Processors that have already been assigned to that Provider or agents residing on the same platform. This inter-platform communication and agent reassignment overloads. In the brokered architecture, all agent assignments are performed by the Broker, which doubles as the Performance Manager.

**Group Halting/Resumption:** Group Halting applies where a Provider group is processing jobs quicker than the agents assigned to it. In such a scenario, the Providers build up large queues of jobs to be processed, while continuing to process jobs themselves.

To avoid this, the Performance Manager instructs the Provider group to temporarily cease processing documents, in order to allow other agent groups to clear the backlog. This is based on a threshold for the number of jobs that should be allowed to be outstanding (this threshold is currently based on the ratio of the number of outstanding jobs to the number available Processors). The Providers are instructed to resume once the backlog has been cleared.

**Agent Creation:** Initially, the system begins with a small agent community. Once the system is running, the Performance Manager incrementally increases the size of the agent community according to its internal model of the current state and performance of the system. The incremental nature of the addition of new agents allows the Performance Manager time to ascertain the impact on performance on each addition of an agent.

Based on the number of agents in each group and the outstanding job queue for that group to process, the Performance Manager decides which group will benefit most by the creation of an additional agent and initiates this creation process. There are two constraints that will cause a group not to be selected for the addition of an agent: groups that have been instructed to halt will not have an additional agent created and a new agent cannot cause the number of agents to exceed the number of outstanding jobs to be processed by that group.

**Agent Destruction:** When the system reaches its maximum capacity (i.e. no more agents can be created), the Performance Manager investigates whether it is possible to create free space for new agents by destroying unnecessary agents, or agents from overpopulated groups.

Once the Performance Manager has identified a group from which it wishes to destroy an agent, it selects an agent to be destroyed from that group. In doing

so, it prioritises the destruction of agents that are not currently engaged in a processing task and whose output queues are smallest.

## 4 Experiments and Evaluation

In order to compare the performance of the brokered and broadcast architecture, a number of experiments were run. Each time the system was run, three Data-Gatherers were created, each gathering documents from a different document collection. The collections used were Cranfield, NPL and the 2Gb Web Track collection from the TREC conference.

**Table 1.** Time to index 3,000 documents (in milliseconds)

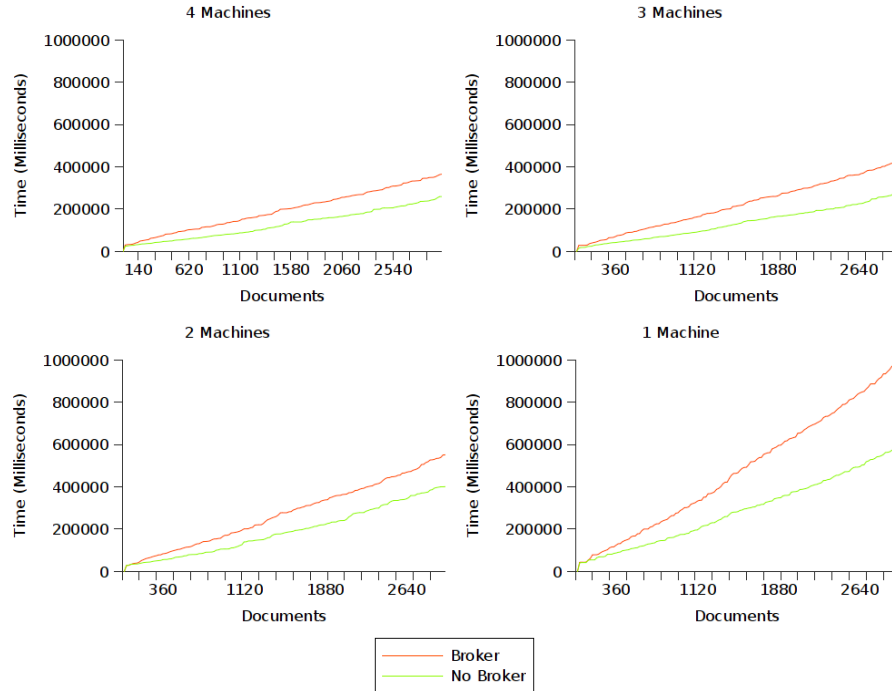
	Broker	Broadcast	% difference
1 machine	987849	585957	-40.68%
2 machines	551573	400895	-27.32%
3 machines	418200	274210	-34.43%
4 machines	365612	259286	-29.08%

The number of machines available to the system was increased so as to evaluate the scalability of each of the two architectures. The creation of agents on any of these platforms was the decision of the Performance Manager, which was present for both architectures. In the brokered architecture, the Performance Manager also took on the role of a Broker. The maximum number of agents that could be created on each platform was manually set at 15. Systems were evaluated by their performance in successfully indexing 3,000 documents. As the document collections contain more than 3,000 documents, in each case all elements of the system were still running on the termination of the experiment.

The results of running these experiments on between 1 and 4 machines are displayed in Table 1 and Figure 1. In each case, the system was run three times and all values used are the average of these three runs. Figure 1 shows the number of documents indexed by each system configuration over time. Table 1 shows the overall time taken to complete the indexing of the 3,000 documents.

For each configuration of the system, the broadcast architecture outperformed the brokered architecture to a large degree. The extent of this difference was always in excess of 27%. The addition of an extra machine reduced the total processing time in each case, as expected. An interesting feature to note is that the impact of the addition of the fourth machine was not as dramatic as that of the second or third machines. This would suggest that the Performance Manager is not currently making full use of the resources being made available to it. Although this is an interesting observation, it does not affect the comparison between the brokered and broadcast architectures, as the same performance management strategy was used in both.

Fig. 1. Document Indexing Speed



## 5 Conclusions and Future Work

This paper has presented the HOTAIR architecture, a scalable agent-based information retrieval system. Specifically, it has introduced two approaches to the self-configuration of this architecture. The first approach, which is described in Section 3.2, follows common practice within the agent community through the modelling of the three core services (data gathering, translating, and indexing) as agents, and the introduction of a Broker agent that acts as both a directory facilitator and a performance manager. The Broker maintains a model of which core agents each document agent is associated with, and periodically analyses that model in order to evaluate how effectively the system is performing.

The second architecture, described in Section 3.3, allowed each core agent a much greater degree of autonomy, in that it could use its own internal model of the system to effect its own behaviour. This removed the central point of failure represented by the Broker, as the system could agent failures while still successfully carrying out the indexing of documents.

The Performance Manager is an important feature of both architectures. It builds an extended model of the system state that includes location and assignment information. The Performance Manager performs periodic analysis of the

current system state, modifying the assignments between core agents in order to improve the performance of the system. It is also empowered with a number of other actions it can take to optimise system performance.

Experimental results show that the broadcast architecture, in addition to the advantages it provides in terms of robustness, results in superior system performance due to the lower communication overhead. The reduction in the time taken to perform document indexing was always in excess of 27%.

Future work will involve further investigation on the scalability of the system on larger clusters of machines. Additionally, we aim to focus on the development of appropriate self-optimisation algorithms, which will make best use of the available performance management actions (outlined in Section 3.4) so as to increase system throughput. Another key element of autonomic computing which we aim to address is self-healing, which should ensure that the system should be capable of tolerating and recovering from agent failures, while ensuring that all documents are successfully indexed.

## References

1. Horn, P.: Autonomic computing: IBM's perspective on the state of information technology. Manifesto, IBM Research (October 2001)
2. Ganek, A.G., Corbi, T.A.: The dawning of the autonomic computing era. *IBM Systems Journal* **42**(1) (2003) 5–18
3. Sterritt, R., Bustard, D.W.: Autonomic computing - A means of achieving dependability? In: *ECBS, IEEE Computer Society* (2003) 247–251
4. Hanson, J.E., Whalley, I., Chess, D.M., Kephart, J.O.: An architectural approach to autonomic computing. In: *ICAC '04: Proceedings of the First International Conference on Autonomic Computing (ICAC'04)*, Washington, DC, USA, IEEE Computer Society (2004) 2–9
5. Lazarou, V., Clark, K.: A multi-agent system for distributed information retrieval on the world wide web (1997)
6. McDermott, P., O'Riordan, C.: A system for multi-agent information retrieval. In: *AICS '02: Proceedings of the 13th Irish International Conference on Artificial Intelligence and Cognitive Science*, London, UK, Springer-Verlag (2002) 70–77
7. Odubiyi, J.B., Kocur, D.J., Weinstein, S.M., Wakim, N., Srivastava, S., Gokey, C., Graham, J.: Saire-a scalable agent-based information retrieval engine. In: *AGENTS '97: Proceedings of the first international conference on Autonomous agents*, New York, NY, USA, ACM Press (1997) 292–299
8. Peng, L., Collier, R., Mur, A., Lillis, D., Toolan, F., Dunnion, J.: A self-configuring agent-based document indexing system. In: *Proceedings of the 4th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2005)*, Budapest, Hungary, Springer-Verlag GmbH (2005)
9. Ross, R., Collier, R., O'Hare, G.: Af-apl bridging principles & practice in agent oriented languages. In: *Proceedings of the First International Workshop on Programming Multiagent Systems, Languages and Tools - PROMAS 2004*, New York, USA (2004)
10. Collier, R., O'Hare, G., Lowen, T., Rooney, C.: Beyond prototyping in the factory of agents. In: *Proceedings of the 3rd International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2003)*, Prague, Czech Republic, Springer-Verlag GmbH (2003) 383

11. Collier, R.: Agent Factory: A Framework for the Engineering of Agent-Oriented Applications. PhD thesis, University College Dublin (2001)
12. Decker, K., Sycara, K., Williamson, M.: Middle-agents for the internet. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence, Nagoya, Japan (1997)
13. Kolp, M., Do, T.T., Faulkner, S., Hoang, T.H.: Introspecting Agent-Oriented Design Patterns. In: Handbook of Software Engineering and Knowledge Engineering. Volume 3: Recent Advances. World Scientific (2005) 105–134
14. Kumar, S., Cohen, P.R.: Towards a fault-tolerant multi-agent system architecture. In: AGENTS '00: Proceedings of the fourth international conference on Autonomous agents, New York, NY, USA, ACM Press (2000) 459–466 Reference for robust Broker-based system.