Monitor Distributed Image Recognition Environment Using Intelligent Agents

Aizhong Lin and Xianjian He
Faculty of Information Technology,
University of Technology, Sydney,
POBox 123, Broadway
NSW 2007, Australia

Abstract

Distributed computing is applied to image recognition by decomposing a large image processing task to small sub tasks and then those sub tasks are allocated to computers that are connected to a computer network to run. This paper describes a real-time situation monitor --- intelligent agent --- that is installed in a image processing computer to detect the real-time environment of the computer and report the situations to the computer --- host --- which is responsible for decomposition of image processing tasks and integration of the results. By using the reports of intelligent agents, the host can make better decisions to choose suitable computers to run sub-tasks. In this way, the performance of image recognition can be increased.

Keywords Distributed Image Recognition, Intelligent Agent

1. Introduction

Image recognition [HWH01] usually means processing huge amount of image data. Image data process needs large amount of computing time and memory. Single computer or CPU (Central Process Unit) is inadequate to cope with it. The technology of parallel computing or distributed computing is applied to image data processing. By using each of those technologies, an image processing task can be firstly decomposed into small sub-tasks, then those smaller tasks can be allocated to different CPUs or computers to perform, and finally the results of all sub tasks are integrated into a final result.

Parallel computing [SM95] allocates sub-tasks of an image recognition task into different CPUs via specific communication architecture. Only the professional image processing company such as a meteorological department or a satellite image processing department is affordable to equip those specific computers such as multiple CPU mainframes. Parallel computing is not suitable to be applied to desktop or laptop computers that have only one CPU.

Distributed computing [HMV97] allocates small tasks into different computers including desktops or laptops via computer network. Distributed computing is getting more and more attention in image recognition area due to computer network, especially Internet, is available in a department, an organization, even almost every where in the world. Most of image processing research groups in small companies or university faculties are equipped with numbers of personal computers. The image processing sub-tasks can be allocated to those existing personal computers even they are used for other purposes such as word processing or scientific computing simultaneously.

To assign image processing tasks into general purpose personal computers such as desktops or laptops, those computers have firstly to be connected to computer network so that they can share CPU time. Secondly, the host of the image processing task (a *host* is a computer in which a software component is responsible for the

task decomposition and results integration) has to know other computers' runtime situations such as CPU performance, available memory space, and so on, so the host can allocate its tasks to better computers that have more CPU time than others and have more free memory than others to do other tasks. And finally the image processing tasks are performed in those computers that have software image processing components.

As computer network, especially Internet, is available in almost every where in the world, it is no longer a problem to connect any computer to a network. We are not going to discuss software components of distributed image processing in this paper. We focus on the real-time environment monitors that detect the real-time situations of general purpose personal computers and report the situations to image processing task hosts so that the hosts can make better decisions to choose suitable computers to run sub-tasks. In this way, the performance of image data processing can be increased.

Intelligent agents [MP93] are computer components that situate in specific environments and that are capable of autonomous and flexible actions to respond the changes of the environment. Intelligent agents, because of their properties of automatic environment detection and autonomous actions, are chosen to be the runtime situation monitors. In this paper, we firstly introduce the multi-agent monitoring architecture of distributed image recognition. Then the architecture and functions of our agent are described. Next, we describe the functions of managing agent reports. And finally we introduce the strategy of choosing better computers to assign small image processing task.

2. Multi-agent Monitoring Architecture

Figure 1 is a multi-agent architecture proposed to monitor a distributed image recognition system. Each personal computer in an image recognition research group is installed with an image processing program and a monitoring agent. An agent situating a computer has three tasks. One task is to detect the real-time situation of a computer including its computer performance, its working rates, and its memory availability. Another task of an agent is to monitor the running situation of the image processing component in the computer. And the third task is to communicate with other monitor agents.

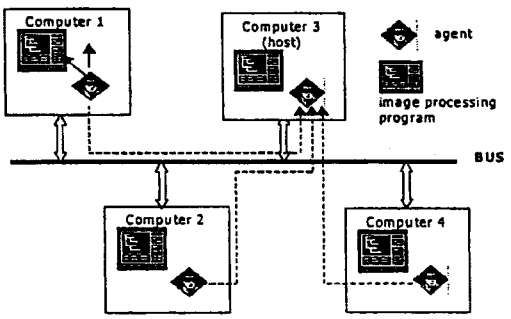

Figure 1: A multi-agent monitoring architecture

An agent has ability to communicate with image processing host agent for it has to report the real-time situations of the computer to the agent in the image host computer. Because any computer could be an image host, any agent should be able to communicate with any other agent. The relationships between those agents are peer-to-peer relationships. When an agent is dynamically set up to an image host monitor, the functions built in this agent that normally is hidden are activated so that this agent has ability to decompose an image task to smaller tasks and allocate all those tasks to other suitable computers. When a client monitoring agent (the agent

situates in a computer that is not currently image processing host) receives a small image processing task, it will activate the image processing program to perform the task.

The logical topology of the multi-agent monitoring system is mesh topology (Figure 2(a)). In this topology, each agent has a channel to communicate with each other agent in the system. This is a pure distributed topology for any disconnection in one channel does not affect any other communication channel. However, when performing an image processing task, this topology is looked like a star topology because only the image host agent needs to communicate with all other selected agents. The image client agent has only to report or respond information to the host agent (Figure 2(b)).

The mesh topology destines each agent in the monitoring system has to be both client and server. This is compatible with the real situation because any computer could be an image processing task host. When a computer becomes an image processing host dynamically, it is a communication server, others are client. In next image processing task, another one may become the host, so communication server keeps changing dynamically. If two image tasks are performed simultaneously, an agent could be a server and a client in the same time.

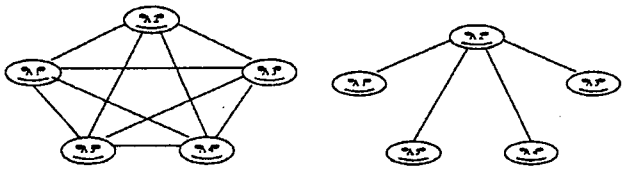

Figure 2 (a): Mesh topology of agent communication
Figure 2(b): The communication situation when performing an image processing task

3. Agent Architecture and Functions

The architecture of our intelligent agent is illustrated with figure 3. Firstly, an agent has functions to monitor three kinds of runtime situations of a computer (environment). The first kind of situation is about the computer runtime general information such as the performance of CPU and the availability of RAM and so on. The second kind of situation is about the distributed image processing component --- process --- that is running in the computer. The information collected from the process includes the process name, CPU time, memory usage and so on. The third kind of situation is the message from other agent(s) because agents communicate with each other via messages. A message is represented by using XML (eXtendible Markup Language) [XML00] and wrapped by using ACL (Agent Communication Language) protocol [FIPA98].

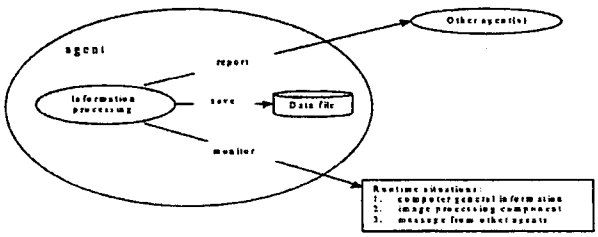

Figure 3: Agent architecture and functions

Secondly, an agent has abilities to handle the runtime information that is collected from those three kinds of runtime situations. More information could be derived according to the collected information. The additional information includes the mean CPU idle time during one hour or a day or a week, the mean

available memory, the mean performance of the image processing component.

Finally, an agent has functions to save the collected information and derived information to a persistent object (a data file) and report the information to its image task host agent if it is an image task client agent. The information is important to a host agent because it is used to allocate current or further image processing task. Additional functions are built in an agent to help users of image task host to decompose an image processing task. This is beyond this paper.

This agent is implemented by using Java language, XML and ACL protocols. Java socket-based network programming technology is used to realize multi-agent communication. Java multithread technology is employed to realize runtime situation monitor. The monitor rate --- times to detect the situations per second --- and the communication rate --- times to send reports per second --- can be set up by users or host agent. Figure 4 is the interface of a monitor agent.

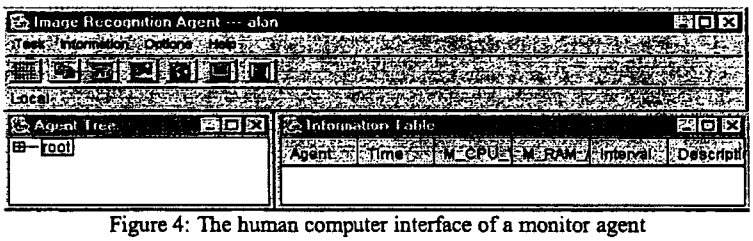

Figure 4: The human computer interface of a monitor agent

4. Multi-agent Communication Message

```
(report
        :sender         alan agent
        :receiver       robert agent
        :language       XML
        :ontology
        :content        ......
)
```

Figure 5: A "report" message wrapped by using ACL

An agent can send or receive four different kinds of messages. Firstly, as an image processing task host, an agent could send a "configure" message to client agent to set up the options of a client agent. For example, a host agent may like the client agent to report its monitor in 30 seconds. Secondly, a host agent could send a "ask" message to a client agent to obtain relevant information. For example, a host only wants the client's mean CPU working rates. Thirdly, a client agent may send a "report" message to a host agent to report all information of general situation and process situation. Finally, a client agent may send "respond" message to reply the host's "ask" message in which only the information asked by the host is included.

A message has a "envelope" of ACL. The envelope includes a primitive, the sender's id, the receiver's id, the type of the language of the message content (default is XML), ontology of the message (default is null), and message content. The message primitive in our agent could be "configure", "ask", "report", and "respond". Figure 5 is a message "report" wrapped by using ACL protocol.

The message contents in an agent are represented by using XML. The XML DTDs of message "configure", "ask", "report" and "respond" are defined in Figure 6(a), 6(b), 6(c), and 6(d) respectively. Where:

- N_T_V: N_T_V is a triple of name, type, and value
- G_info: G_info is the gernral information that includes two elements:
  - M_CPU_W_R: It is the CPU working rate in a period of time. The default time interval is 10 seconds.
  - M_RAM_A_R: It is the RAM available rate in a period of time.
- P_info: P_info is the process information that includes following elements:

- P_name: It is the process name of image processing component
- CPU_time: It is the CPU time of this process
- RAM_usage: It is the memory usage of this process

```
<!ENTITY %types "(int|double|boolean )">

<!ELEMENT configure (N_T_V)*>

<!ELEMENT N_T_V EMPTY>
<!ATTLIST N_T_V
      name (#PCFATA)
      type %types
      value (#PCDATA)>
```

Figure 6(a): DTD of message "configure"

```
<!ELEMENT ask (G_info,P_info)>

<!ELEMENT G_info (Name)*>
<!ELEMENT Name EMPTY>
<!ATTLIST Name
      name (#PCFATA)>

<!ELEMENT P_info (Name)*>
<!ELEMENT Name EMPTY>
<!ATTLIST Name
      name (#PCFATA)>
```

Figure 6(b): DTD of message "ask"

```
<!ENTITY %types "(int|double|string)">

<!ELEMENT report (G_info,P_info)>

<!ELEMENT G_info (N_T_V)*>
<!ELEMENT N_T_V EMPTY>
<!ATTLIST N_T_V
      name (#PCFATA)
      type %types
      value (#PCDATA)
      interval (#PCDATA) >

<!ELEMENT G_info (N_T_V)*>
<!ELEMENT N_T_V EMPTY>
<!ATTLIST N_T_V
      name (#PCFATA)
      type %types
      value (#PCDATA)>
```

Figure 6(c): DTD of message "report"

```
<!ENTITY %types "(int|double|string)">

<!ELEMENT report (G_info,P_info)>

<!ELEMENT G_info (N_T_V)*>
<!ELEMENT N_T_V EMPTY>
<!ATTLIST N_T_V
      name (#PCFATA)
      type %types
      value (#PCDATA)
      interval (#PCDATA) >

<!ELEMENT G_info (N_T_V)*>
<!ELEMENT N_T_V EMPTY>
<!ATTLIST N_T_V
      name (#PCFATA)
      type %types
      value (#PCDATA)>
```

Figure 6(d): DTD of message "respond"

```
<?xml version="1.0" ?>
<!DOCTYPE agent "report.dtd">

<report>
      <G_info>
            <N_T_V name="M_CPU_W_R" type="double" value="0.35" interval="60s">
            <N_T_V name="M_RAM_A_R" type="double" value="0.74" interval="3600s">
            <N_T_V name="M_RAM_A_R" type="double" value="0.74" interval="3600s">
      </G_info>

      <P_info>
            <N_T_V name="P_name" type="string" value="image_processing_1">
            <N_T_V name="CPU_time" type="string" value="0.28h">
            <N_T_V name="RAM_usage" type="string" value="1120K">
      </P_info>
</report>
```

Figure 7: A typical XML of message report

For example, based on the report.dtd, a typical "report" message could be represented as in Figure 7.

5. General and Process Information Management

After the general and process information is detected from the runtime environment by an agent, the information is firstly saved in the agent data file persistently. From the runtime information, more information such as Mean_CPU_Working_Rate in different interval can be derived. The information may be reported to host agent from client agent immediately or stay at the client agent until the host agent asks for it. Which kind of action --- report or stay --- should be taken depends on the agent options set by host agent. If an agent is currently a host agent, it can save and manage the information that came from the all its client agents.

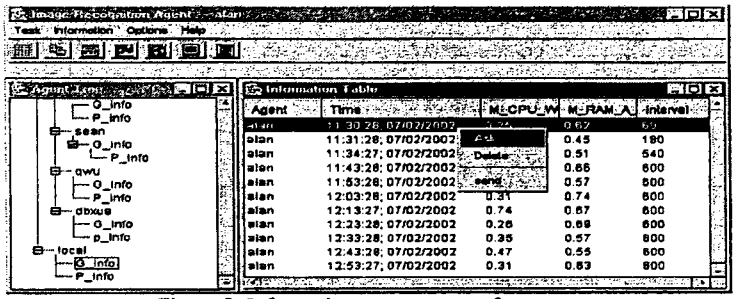

Figure 8: Information management of an agent

Each agent provides management functions to manage its general and process information and other agents' general and process information when it is a host agent. Those functions include view, delete, and send (Figure 8).

- Ask: request the current situations of a client agent
- Delete: if the information is no longer needed, it can be deleted by the agent user
- Send: the information can be manually sent to any host agent

6. Task Allocation Strategy

The goal that we use agents to monitor distributed image processing computers is to choose "better" computer to run the image processing sub tasks. Because those computers may have other tasks to perform, it is not a good choice and the image processing performance may be low if we allocate an image sub task to a heavy running computer. Allocating a task to a suitable computer could be done by an agent user manually, however, it can be done by an agent automatically. If we expect an agent (host agent) to allocate the tasks, the allocating strategy should be told to the agent by the user.

Task allocating strategies are desired in each agent and are applied to the task allocation. A task allocating strategy is a set of rules that are used for host agent to choose suitable computers to run image processing sub tasks according to the information collected by agents.

```
//A task is allocated to a computer with the maximum CPU idle time and its //memory
available
rule 1:
  while (allocating)
  {
        //calculate M_CPU_W_R and M_RAM_A_R from agent i by setting interval
        for(int i=0; i<NumberOfAgent; i++)
        {
          agent.setMCPUWR(CalculateMCPUWR(interval)); //interval is integer
          agent.setMRAMAR(CalculateMRAMAR(interval));  //interval is integer
          agent.setGCR(agent.getMRAMAR() * (1 - agent.getMCPUWR()));
        }

        //if we have to allocate k tasks
        int count = k;
        while(count < 1)
        {
          int j = findAgentThatHasMax_GCR();
          allocating(task(count), computer(j));
          drop(agent(j));
          count--;
        }
  }
    ......

//A task is allocated to an agent with the maximum task completion rate and //memory
usage
rule 2:
  while (allocating)
  {
        //calculate M_CPU_W_R and M_RAM_A_R from agent i by setting interval
```

```
    for(int i=0; i<NumberOfAgent; i++)
    {
      agent.setPCPUT(CalculatePCPUT(k));//k is the number of previous process
      agent.setPRAMU(CalculatePRAMU(k));//k is the number of previous process

      agent.setPCR(agent.getPCPUT() * agent.getPRAMU();
    }

    //if we have to allocate k tasks
    int count = k;
    while(count < 1)
    {
      int j = findAgentThatHasMax_PCR();
      allocating(task(count), computer(j));
      drop(agent(j));
      count--;
    }
}
```

7. Future Work

The multi-agent system is built and installed in a research laboratory for distributed image recognition. They are running well and report the client agents' general information and process information to a task host agent. However, the information collected to a host is currently used by the human users manually, i.e., human users use the information manually to choose suitable computers to run sub tasks. Because it has not been sufficiently used for automatic task allocation, great improvement of image recognition task has not been reached. Our feature work is to sufficiently use the information for automatic task allocating so that to greatly increase the image recognition performance.

References

[SM95] V. P. Srini, DFS-SuperMPx: Low-cost Parallel Processing System for Machine Vision and Image Processing. In V. Malyshkin (ed.), Parallel Computing Technologies. Proceedings of the 3th International Conference, Lect. Notes in Comp. Sci., Vol. 964, Springer, 1995, pp. 356-369

[HMV97] K.A.Hawick, K.J.Maciunas and F.A.Vaughan. DISCWorld: A Distributed Information Systems Control System for High Performance Computing Resources. DHPC-014, 1997. Department of Computing Science, University of Adelaide.

[HWH01] X. He, Q. Wu and T. Hintz. Spiral Object Recognition on Clusters. Proceedings CISST'2001. Las Vegas, June 2001, pp605-611.

[MP93] Jörg P. Müller and Markus Pischel. The Agent Architecture InteRRaP: Concept and Application. Research Report, Deutsches Forschungszentrum für Künstliche Intelligenz, Number RR-93-26, p. 99, 1993.

[XML00] Extensible Markup Language (XML) 1.0 (Second Edition). http://www.w3.org/TR/REC-xml. W3C Recommendation 6 October 2000

[FIPA98] FIPA specification: Agent Communication Language. http://www.fipa.org/specs/fipa00003/OC00003A.html. Fundamental Intelligent Physical Agent. 1998.