(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
21 December 2007 (21.12.2007)

PCT

(10) International Publication Number
WO 2007/144891 A1

(51) International Patent Classification:
G06F 9/44 (2006.01)

(21) International Application Number:
PCT/IL2007/000727

(22) International Filing Date: 14 June 2007 (14.06.2007)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
176348 15 June 2006 (15.06.2006) IL

(71) Applicant *(for all designated States except US)*: **XOREAX LTD.** [IL/IL]; 9 Shprintsak Street, 64738 Tel Aviv (IL).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **SHAHAM, Uri** [IL/IL]; 54 Ben Gurion Street, 45200 Hod Hasharon (IL). **MISHOL, Uri** [IL/IL]; 9 Shprintsak Street, 64738 Tel Aviv (IL).

(74) Agents: **LUZZATTO, Kfir** et al.; P.O. Box 5352, 84152 Beer Sheva (IL).

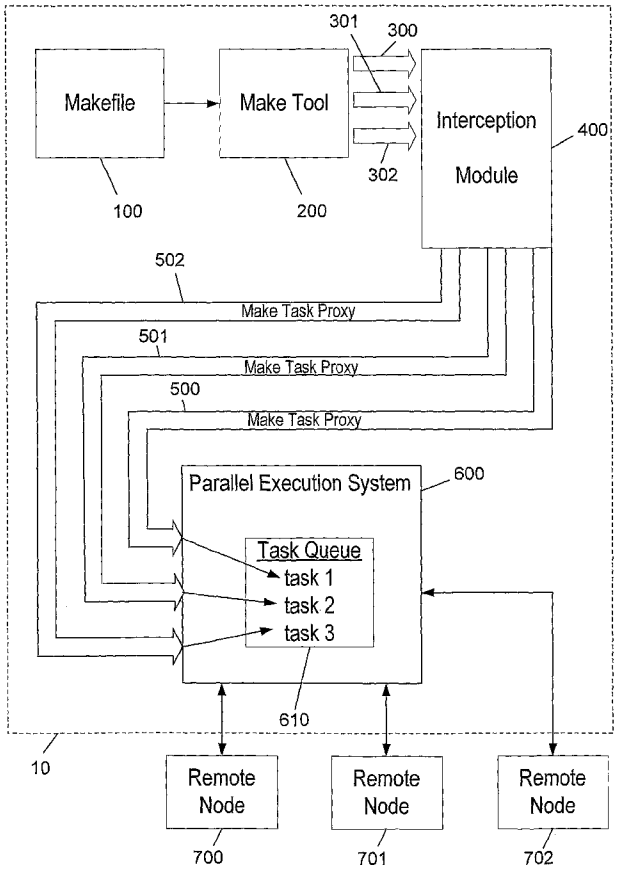
(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, MT, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

[Continued on next page]

(54) **Title:** A METHOD FOR THE DISTRIBUTION OF SOFTWARE PROCESSES TO A PLURALITY OF COMPUTERS



(57) **Abstract:** Method for distributing tasks of a Make tool to a plurality of interlinked computers, according to which the Make tool is executed to operate in local parallel mode and a request for a child process creation of the Make tool is re-directed to a Parallel Execution System, for distribution. At least one task listed in the Parallel Execution System is distributed to one of the interlinked computers. At least one indication of the task execution completion is received by the Parallel Execution System, from at least one of the interlinked computers and then control is returned back to the Make tool.

WO 2007/144891 A1

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

A METHOD FOR THE DISTRIBUTION OF SOFTWARE PROCESSES TO A PLURALITY OF COMPUTERS

Field of the Invention

The present invention relates to the field of distributed processes over a plurality of computers, connected to a data network. More particularly, the invention relates to a method for parallel execution of a complex computer process such as building source code files into a program, data file generation, complex data processing, QA script execution and other examples, using a number of interlinked computers.

Background of the invention

In recent years, computing jobs related to the development of computer software have grown drastically in size and complexity. Typical software development environments demand, in addition to the ongoing coding process, the ability to regularly execute complex computerized processes for supporting the development process. These typically include generation of various data files, quality assurance tests and "code builds" which translate source code files into executable machine code. Many of these processes can be time-consuming and, when performed regularly, may become a major bottleneck in the software development process.

Fortunately, these processes may be automated with software engineering tools such as the "Make" tool. The Make tool works off a file called the "Makefile" which lists in a structured manner the dependencies of the files involved in the process. The Makefile may also define the commands required to"build" these files. Each entry in the Makefile is a rule expressing the dependencies and the commands needed to build or Make a certain object. Several different "Make" tools exist today, and they differ in the Makefile format they use, in their host platform and in the features they offer.

One of the common examples of a complex software development process is the "Code Build" process. Some of the programs used today are built from numerous source code files, each compiled separately and "linked" with other files to form a final executable or library file. In a typical "build" process, the source code files are first compiled into object files and the object files are then linked together with or without additional files, forming executable files, otherwise known as an executable program. Some files generated throughout the code build process "depend" on other files, meaning that the dependant files include or use parts of other files. In other words, if a file is modified (either manually or as a result of a build process), all of its dependant files require re-building in order for these files to be "up-to-date" on any changes made.

For example, the file final.exe is dependent on an object file code.o and code.o is dependant on a source code file code.cc, whereas code.cc has been modified. During the build process, the Make tool detects that code.cc has been modified and it will re-build code.o and final.exe accordingly as both depend directly or indirectly on code.cc. Nevertheless, typical software projects are far more complex than the representing example and even a modest-size project can have multiple dependencies relating to thousands of files, resulting in an extremely complex dependency structure. In addition, Makefiles can be arranged in a hierarchical structure with higher-level Makefiles invoking lower-level Makefiles to build pieces of the program, adding additional complexity to the build process.

As mentioned before, Make can operate incrementally, only rebuilding files that have changed and, consequently, files that depend on the rebuilt files. Thus in principle it should be possible to build a very large program relatively quickly if only a few files are changed, since most of the files will be considered "up-to-date". In practice, though, there are many times when large portions of a program must be rebuilt, especially when a file, that many other files depend on, has been changed. Integration points, where developers update all the recent changes to the shared repository, typically result in massive recompilation, as well. Furthermore, most organizations support multiple platforms and versions, adding a multiplicative factor to the above building times. Long build times have a

high cost for companies where software development is time-critical. They affect not only engineering productivity and release schedules, but also software quality and overall corporate agility. When a developer makes a change to a source code file it may take more than a full day before the developer can tell whether the change caused a problem.

As mentioned before, Make tools may also be used for purposes other than program building.

One of the solutions used today in order to improve the performance and reduce the processing time of the Make tools is to take advantage of a multiprocessor computer that can effectively perform several tasks in parallel. Most Make tools feature a "parallel" mode, which allows the process of several files simultaneously, thus saving precious time. However, multiprocessor computers typically have only 2-8 CPUs, which limits the potential speedup, and larger-scale multiprocessor computers having as many as 32 or 64 CPUs are quite expensive. A few operating systems allow "clustering" of several machines which act together as a single multiprocessor machine, allowing additional speedup. However, this option is limited to specific operating systems, and it usually requires special configuration and additional expenses.

A different approach used by some software tools is to execute a Make process in parallel, over several independent computers connected to a local network. With this method, the software tool parses the makefile contents to determine the elements described in the makefile and the dependencies between them, and then uses a distributed architecture to execute these files on remote computers in parallel and collecting the outputs from these computers. Although this approach is more effective than others, the fact that it incorporates parsing of the Makefile contents raises two major limitations- firstly, Makefile parsing is inherently prone to software bugs and is intolerant to changes in the Makefile syntax. Secondly, Makefile parsing requires separate implementations for every Makefile format, making existing solutions unavailable to new or less popular Make tools for which a parsing solution has not yet been implemented.

WO 2004/088510 describes a method for analyzing the Makefile and parsing its commands into individual components, where each component is executed on a different machine. Thus the Makefile commands may be run in parallel on a number of connected independent machines. However, this approach is totally reliant on the ability of correctly analyzing the Makefile. Any change in the Makefile format, such as a new version release, requires updating the present tools to cope with the new format.

It is an object of the present invention to provide a method for running Make tools in parallel, using interlinked individual computers, in a manner which is indifferent to the Make tool or to Makefile format used.

It is another object of the present invention to provide a method for parallel building using interlinked individual computers, even where there is no native operating system or hardware support for parallel execution.

It is still another object of the present invention to provide a method for parallel processing using individual machines, which is less expensive, faster, more reliable, and more effective than the described previous prior arts.

Other objects and advantages of the invention will become apparent as the description proceeds.

Summary of the Invention

The present invention relates to a method for distributing tasks of a Make tool to a plurality of interlinked computers. The Make tool is executed to operate in local parallel mode and a request for a child process creation of the Make tool is re-directed to a Parallel Execution System, for distribution. At least one task listed in the Parallel Execution System is distributed to one of the interlinked computers. At least one indication of

the task execution completion is received by the Parallel Execution System, from at least one of the interlinked computers and then control is returned back to the Make tool.

Preferably, the re-directing is performed automatically by pre-installed software.

Preferably, the automatic re-directing is done by copying aside assembly instructions at the beginning of at least one function code in the Make tool process, relevant to the request for child process creation, and replacing them with a new instruction, for re-directing execution to a code generated by the pre-installed software.

Alternatively, the automatic re-directing may be done by creating a modified copy of the Make tool executable, where the modified copy comprises at least one modified name of a system library relevant to the request for child process creation in the Make tool import table, for loading compatible libraries instead.

The automatic re-directing may also be done by adding to the local file system at least one library having the same file name as a system library relevant to the request of child process creation before the system directories in the Make tool process search path.

Preferably, the automatic re-directing is done by creating a modified copy of the make-process executable before execution and adding new functions that replace the system functions.

Preferably, the re-directing is performed by the Make tool as a result of the Makefile(s) modification(s).

Preferably, the indication is an Exit code.

Preferably, the indication is an Output text(s) or output file(s) sent by the interconnected computer.

Brief Description of the Drawings

In the drawings:

- Fig. 1 is a block diagram generally illustrating the method for parallel processing, using a plurality of interlinked computers, according to one of the embodiments;
- Fig. 2 is a block diagram generally illustrating another embodiment of the invention for parallel processing using a number of interlinked computers; and
- Fig. 3 is a block diagram generally illustrating the Parallel Execution System according to an embodiment of the invention.

Detailed Description of Preferred Embodiments

For the sake of brevity the following terms are defined explicitly:

- Task - a computational process, such as the compilation of a source code file.
- Command line (Task command line) – text, string, comprising: a command or executable path, and optionally additional arguments, for interpretation and execution by an operating system.
- Make process - software process, for executing tasks in a controlled order.
- Makefile - a file, or other forms of electronic storage record, describing a Make process, or a part of it, listing for each task, in a structured manner, one or more of the following: the Task's command line, its input/output files, its dependency relationship with other tasks, and additional information.
- Make - a software tool for automating execution of Make processes. Make parses command line and input/output file information stored in one or more MakeFile(s), and uses this information to execute the Tasks required to complete the software process in the correct order.
- Make tool - a generic name used hereinafter for software tools having the same essential function of Make, such as JAM, NANT, ANT, Scons, omake, gmake and other similar tools.

- Local Parallel Mode - a Make tool execution mode, in which two or more Tasks are executed in parallel in order to speed up the Make tool execution process. Primarily used with multiprocessor machines.
- Parallel Execution System (PES) - a software system capable of managing task execution on a group of interlinked computers, including: executing tasks on local or remote computers, returning resulting outputs to requesting process, keeping track of remote computers' availability, handling task assignments, and handling file and environment synchronization issues to ensure correct execution based on the initiating machine's environment and file system.
- Request for child process creation - a request to the operating system made by a software process for the purpose of creating a new "child" software process that will execute a specified command line.

Fig. 1 is a block diagram generally illustrating the method for parallel processing, using a plurality of interlinked computers, according to one of the embodiments. The Make tool 200 is run using local parallel mode, as if running on a multiprocessor server. The Make tool 200 reads the Makefile 100, determines which Tasks need to be executed (either because one or more of the Task's output files is older than one or more of its input files,

or because a full rebuild instruction) and, assuming one or more tasks require execution, attempts to execute tasks by sending requests 300-302 for child process creations addressed to the local Operating System (not shown), where each request contains the task's command line. Since the Make tool 200 is running in the Local Parallel Mode, it may send several requests in parallel for simultaneous execution. Hence, for the sake of brevity, the description deals with an example of three requests 300-302. The interception module 400 is installed on the local machine 10 which is the computer that executes the Make Tool 200. The interception module 400 captures/intercepts child process creation attempts made by Make tool 200 in the local machine 10. Before the child processes are actually created, and instead of creating the child process, it creates a Make Task Proxy process (MTP) 500-502. In other words, the interception module 400 is capable, when required, of re-directing the Make tool 200 child process creation instructions from their intended destination as opposed to the prior art procedure, where the child process creation attempts results in the creation of a child process for executing the task in the local operating system. Each one of the MTP 500-502 corresponds to one of the tasks 300-302 and stores that task's command line accordingly. The MTP 500-502 add their stored command line to the task queue 610 of the Parallel Execution System (PES) 600, and wait for the task to be executed by the PES 600. The PES 600 analyzes the incoming command lines and decides which of the command lines should be executed locally and which of the

command lines may be executed remotely. The command lines selected for local execution are executed on the local machine 10. The command lines selected for remote execution are thus sent by PES 600 to one of the available interlinked computers 700-702, referred to hereinafter as remote nodes, for execution. After one of the remote nodes 700-702 executes a task corresponding to the received command line, it sends back the exit code returned by the task and/or any textual output produced by the task to the PES 600. For example, when remote node 700 executes task 1, it sends back an exit code and/or any textual output produced by task 1. The PES 600 returns the exit code and/or textual output to the MTP 502, which had added the command line to the list, where the MTP 502 returns the exit code and/or textual output to the Make tool 200. Furthermore, the PES 600 may transfer any output files created on the remote node as a result of the task execution to the local machine 10, before returning control to the corresponding MTP 500-502. The Make tool 200 continues creating parallel child process creation attempts using the local parallel mode. Each one of the child process creation attempts is intercepted, sent to PES 600, and its corresponding task is executed either locally or remotely, obliviously of the Make tool 200 version, the Makefile 100 syntax, or its format. All through the software process the Make tool 200 continues executing as if there are multiple processors which are executing the tasks simultaneously, although in truth the tasks are actually executed by a number of computers.

In a multiprocessor server the Make tool's local parallel mode is typically configured to run a maximal number of parallel processes that corresponds with the number of processors on the server. However in one of the embodiments of the system the local parallel mode may be configured to use a number of parallel processes that corresponds to the maximum number of nodes available to PES 600 or higher, as the processes may be queued in the task queue obliviously of the real number of computers or processors processing in parallel.

Several methods exist for allowing the interception module 400 to capture/intercept child process creation attempts made by Make tool 200. However, for the sake of enablement, 3 methods are described herein which are not intended to limit the scope of the invention in any way. The first method requires modification of the make-process address space after it is loaded into memory and before its execution starts. Several assembly instructions at the beginning of every function code relevant to the creation of a child process are copied aside, and replaced with an instruction that redirects execution to code generated by the interception module. This generated code allows the interception module to intercept the child process creation attempts and replace them with Interception Module logic. To resume execution of the original code, the copied assembly instructions are executed elsewhere in memory, and then

execution flow is redirected back to the original function. The second method involves creating a modified copy of the Make tool executable before execution of the make-process. Mainly, import tables of the modified executable are patched, modifying the names of system libraries relevant to the creation of child processes, so compatible interception module libraries are loaded instead. Similarly, placing interception module libraries having the same file name as the system libraries ahead of system directories in the search path is also possible. As a result, the make-process would load the interception module libraries, which enable the Interception Module logic, instead of loading the system libraries. The third method involves creating a modified copy of the make-process executable before execution and adding imported functions from an interception module dynamic-link library that replace imported functions from system dynamic link libraries. Calls related to the creation of child processes, made by the make-process, are consequently handled by the interception module dynamic-link library, which will perform the necessary interception actions, and if necessary, redirect execution back to the original calls.

Prior to applying the first method, interception module code has to be "injected" into the make-process address space, in order to establish the correct locations in the memory of the relevant functions. Although "injection" may be done in many ways, for the sake of brevity 3 alternative

methods are described: (a) *Injecting additional code into the make-process* - This is done by executing the make-process in suspended mode, which effectively means loading the executable into memory without starting actual program execution. Once the process is loaded, a memory block is allocated in its address space by the interception module, and filled with all the additional interception module code that's required to run within the make-process address space ("added code"). The entry point of the make-process is then determined, and its entry code is patched with an instruction that redirects execution to the added code. Execution of the make-process is then resumed, resulting in the patched entry code redirecting execution to the added code. The added code patches, as described, restore the original make-process entry code, and redirect execution back to the entry point. (b) *Adding additional code to the executable code section* - Executable files typically contain several sections, such as code, data, or resources. This method involves creating a modified copy of the make-process executable, while adding additional code to its code section, and modifying its entry point, so execution starts at the newly added code. The newly added code patches the function assembly instructions, as described in the first method, and redirects execution to the original entry point of the make-process. (c) *Injecting a dynamic link library into the make-process* - This is done by executing the make-process in suspended mode, which effectively means loading the executable into memory without starting actual program execution. Once the process is

loaded, a memory block is allocated in its address space by the interception module, and filled with a piece of code ("interception initialization code") that loads a dynamic library that is part of the interception module ("injected library"). The entry point of the make-process is then determined, and its entry code patched with an instruction that redirects execution to the interception initialization code. Execution of the make-process is then resumed, resulting in the patched entry code redirecting execution to the interception initialization code. The initialization code loads the injected library, restores the original make-process entry code, and redirects execution back to the entry point. During loading of the injected library, system functions assembly instructions are patched, as described in the first method. Any combination of the described methods may be used as well.

Fig. 2 is a block diagram generally illustrating another embodiment of the invention for parallel processing using a number of interlinked computers. In this embodiment the Makefile 100 is edited manually before starting the process. The command lines of tasks in the Makefile 100 are manually modified to execute a MTP instead of directly executing the specified command line. For example, the command line:

"Calcdata.exe a.dat/p 40"

is modified to:

"MakeTaskProxy.exe CalcData.exe a.dat/p 40"

(Where "MakeTaskProxy.exe" represents the executable filename of the MTP) thus ensuring that a MTP is created for handling the task. Fortunately, many Makefiles use symbol translation that can simplify the manual modification. The file system path for each executable used to execute tasks is often defined in one location in the Makefile and is given a symbolic name. Changing the path in this single location will cause the change to apply to all Make Tasks using that executable in the Makefile. Similarly to the first embodiment, the Make tool 200 is configured to run using Local Parallel Mode. However, since the command lines in the Makefile 100 direct to the MTP executable file, the Make tool 200 requests for child process creation are actually requests for the creation of a MTP process. MTP's 500-502 are thus created, where each MTP stores a command line corresponding to the original command line before modification. In the example, "Calcdata.exe a.dat/p 40" is the original command line. For the sake of brevity the description deals with an example of only three requests 500-502, although in practice many more requests may be present. Each of the MTP's 500-502, once created, adds its command line to the task queue 610 of PES 600. The PES 600 analyzes the incoming command lines and decides which of the command lines

should be executed locally and which of the command lines may be executed remotely. The command lines selected for local execution are executed on the local machine 10. The command lines selected for remote execution are thus sent by PES 600 to one of the available remote nodes 700-702, for execution. After one of the remote nodes 700-702 executes a task corresponding to the received command line, it sends back the exit code returned by the task and/or as any textual output produced by the task to the PES 600. For example, when remote node 700 executes task 1, it sends back an exit code and/or any textual output produced by task 1. The PES 600 returns the exit code and/or textual output to the MTP 502, which had added the command line to the list, where the MTP 502 returns the exit code and/or textual output to the Make tool 200. Furthermore, the PES 600 may transfer any output files created on the remote node as a result of the task execution, to the local machine 10, before returning control to the corresponding MTP 500-502. The Make tool 200 continues creating parallel MTP using the local parallel mode. All through the software process the Make tool 200 continues executing as if there are multiple processors which are executing the tasks simultaneously, although in truth the tasks are actually executed by a number of computers.

For the sake of brevity, an embodiment of a PES is described and illustrated in Fig. 3, although many PES embodiments are possible for

carrying out the invention. As described, PES 600 has a task queue 610 containing parallel tasks for execution. New command lines may be added to the task queue 610, as described above. PES 600 may use the configuration file 620 to determine which of the incoming command lines should be executed locally and which of the command lines may be executed remotely. For example, one of the rules of configuration file 620 may suggest that any command line describing a task that requires use of a hardware component only available on the local machine should always be processed locally. Once PES 600 decides that a command line should be executed locally, the task described by that command line is executed on the local machine as indicated in Local Task Execution 650. The Local Task Execution 650 may use the File system 630 to read and write files accessed by the task. After executing the local task, the Local Task Execution 650 returns an exit code and/or textual output, which are returned to the MTP which added the task to the task queue 610. In addition, the Local Task Execution 650 may store the output files created by the task execution process on the File system 630. When PES 600 decides that a task may be executed remotely, it sends the task through interface 640 to one of the available remote nodes 700-702. For example, if remote node 700 is available, the task is sent to remote node 700 where the task is received by Client PES 710, which executes the task using Client Task Execution 750. If the sent task requires input files from the File system 630, PES 600 may also transfer the input files from File

system 630 to the client PES 710 of remote node 700 for executing the task. After remote node 700 completes the task execution, the exit code and/or output text are sent back to PES 600. If any output files were generated, they too are sent to PES 600, which will store the output file in the local File system 630. The exit code and possibly output text are finally sent to the MTP that added the task to the list 610. As understood, the other Remote nodes 701-702 function in a similar manner.

As understood, other PES embodiments are possible. For example in one of the embodiments the remote nodes have direct access to the local File system, whereas the remote nodes may read or write a file in the File system during task execution. In another embodiment the File system is held on a remote computer accessible to all participating computers.

While some embodiments of the invention have been described by way of illustration, it will be apparent that the invention can be carried into practice with many modifications, variations and adaptations, and with the use of numerous equivalents or alternative solutions that are within the scope of persons skilled in the art, without departing from the spirit of the invention or exceeding the scope of the claims.

CLAIMS

1. A method for distributing tasks of a Make tool to a plurality of interlinked computers comprising the steps of:

a. executing said Make tool to operate in local parallel mode;

b. re-directing at least one request for a child process creation of said Make tool to a Parallel Execution System, for distribution;

c. distributing at least one task listed in said Parallel Execution System to one of said interlinked computers;

d. receiving, by said Parallel Execution System, at least one indication of said task execution completion from at least one of said interlinked computers; and

e. returning control to said Make tool.

2. A method according to claim 1, where the re-directing is performed automatically by pre-installed software.

3. A method according to claim 2, where the automatic re-directing is done by copying aside assembly instructions at the beginning of at least one function code in the Make tool process, relevant to the request for child process creation, and replacing them with a new instruction, for re-directing execution to a code generated by the pre-installed software.

4. A method according to claim 2, where the automatic re-directing is done by creating a modified copy of the Make tool executable, where said modified copy comprises at least one modified name of a system library relevant to the request for child process creation in the Make tool import table, for loading compatible libraries instead.

5. A method according to claim 2, where the automatic re-directing is done by adding to the local file system at least one library having the same file name as a system library relevant to the request of child process creation before the system directories in the Make tool process search path.

6. A method according to claim 2, where the automatic re-directing is done by creating a modified copy of the make-process executable before execution and adding new functions that replace the system functions.

7. A method according to claim 1, where the re-directing is performed by the Make tool as a result of the Makefile(s) modification(s).

8. A method according to claim 1, where the indication is an Exit code.

9. A method according to claim 1, where the indication is an Output text(s) or output file(s) sent by the interconnected computer.

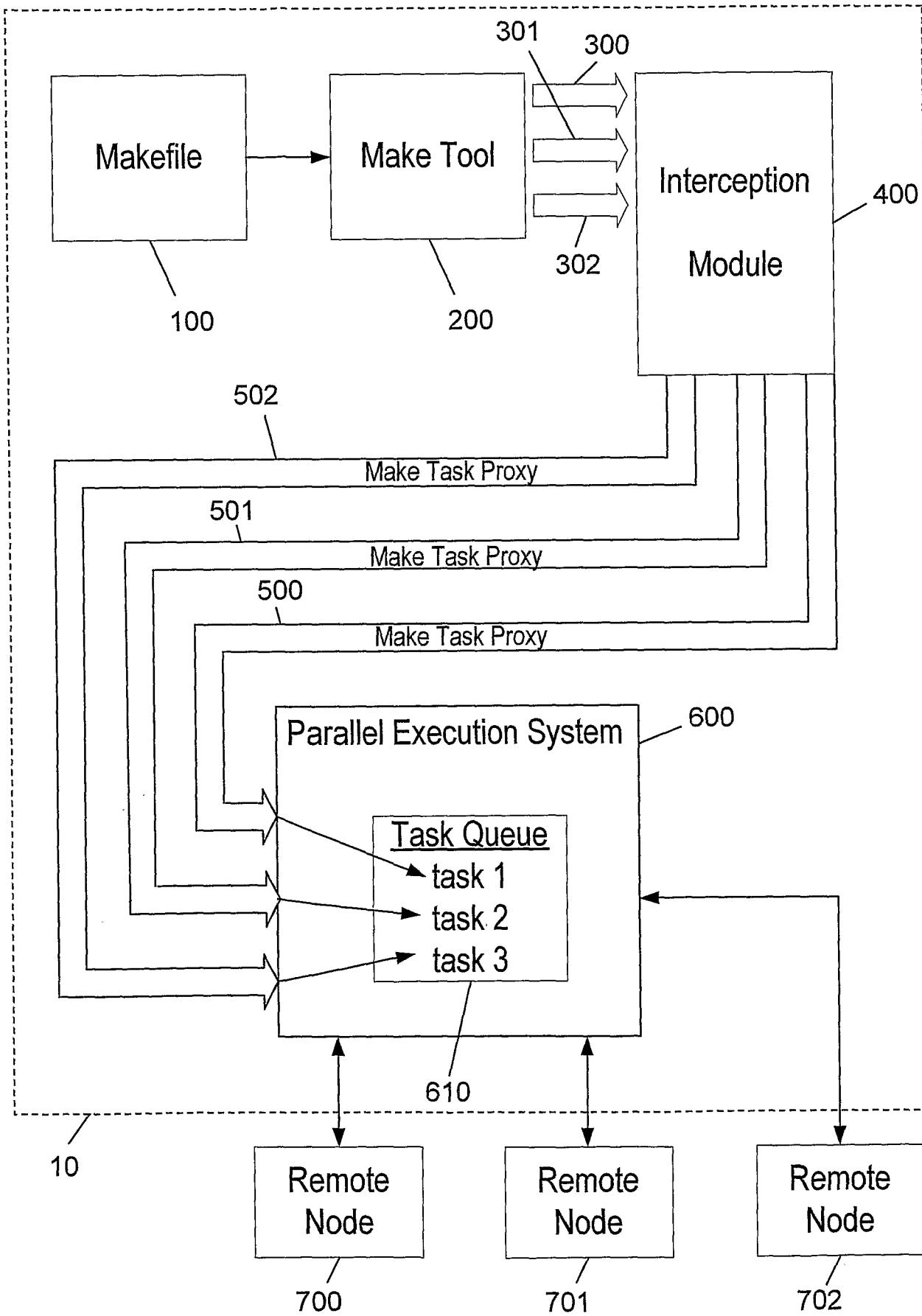

301
300
Makefile
Make Tool
Interception
Module
400
302
100
200
502
Make Task Proxy
501
Make Task Proxy
500
Make Task Proxy
Parallel Execution System
600
Task Queue
task 1
task 2
task 3
610
10
Remote
Node
Remote
Node
Remote
Node
700
701
702

Fig. 1

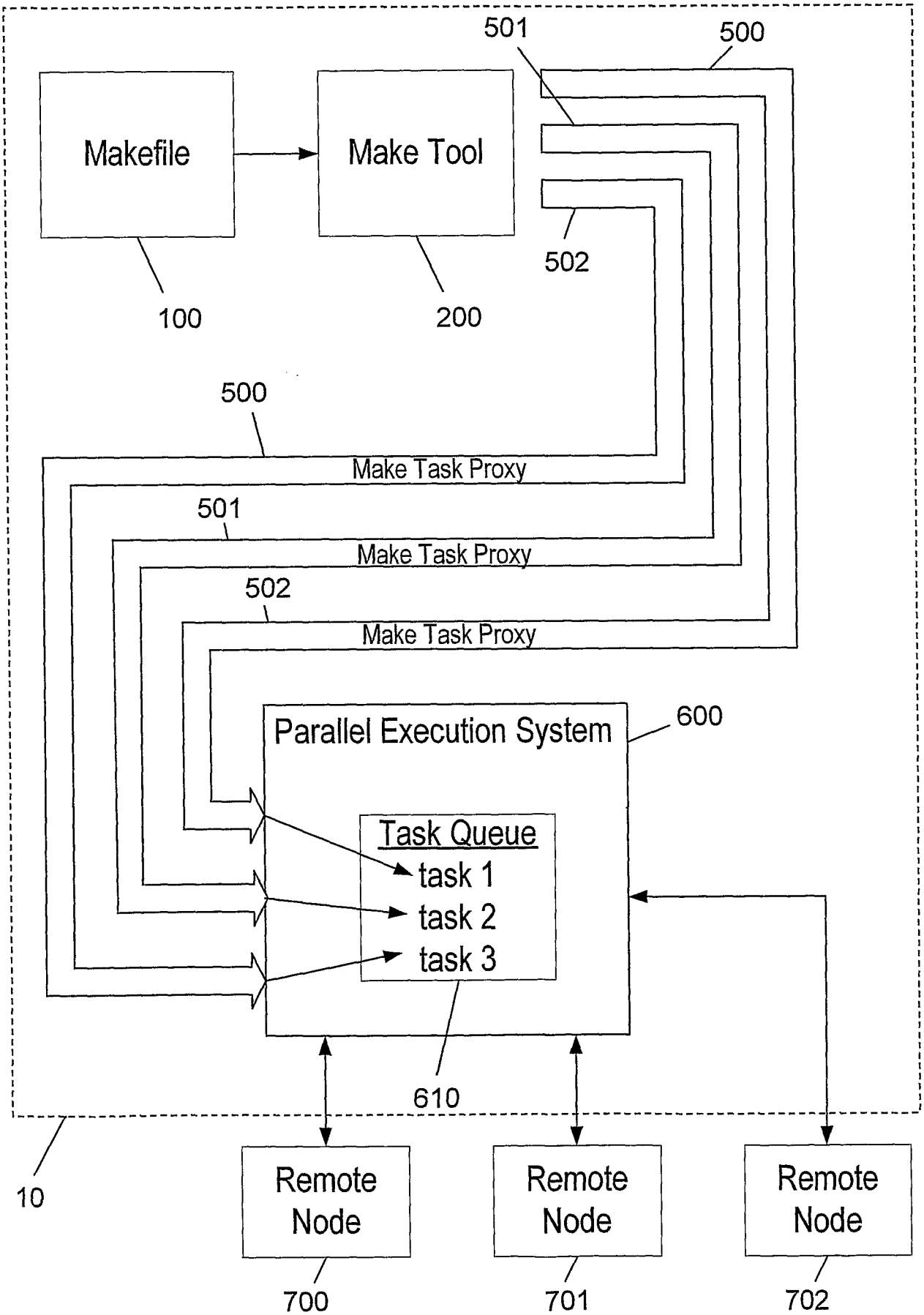

501
500
Makefile
Make Tool
502
100
200
500
Make Task Proxy
501
Make Task Proxy
502
Make Task Proxy
Parallel Execution System
600
Task Queue
task 1
task 2
task 3
610
10
Remote Node
Remote Node
Remote Node
700
701
702

Fig. 2

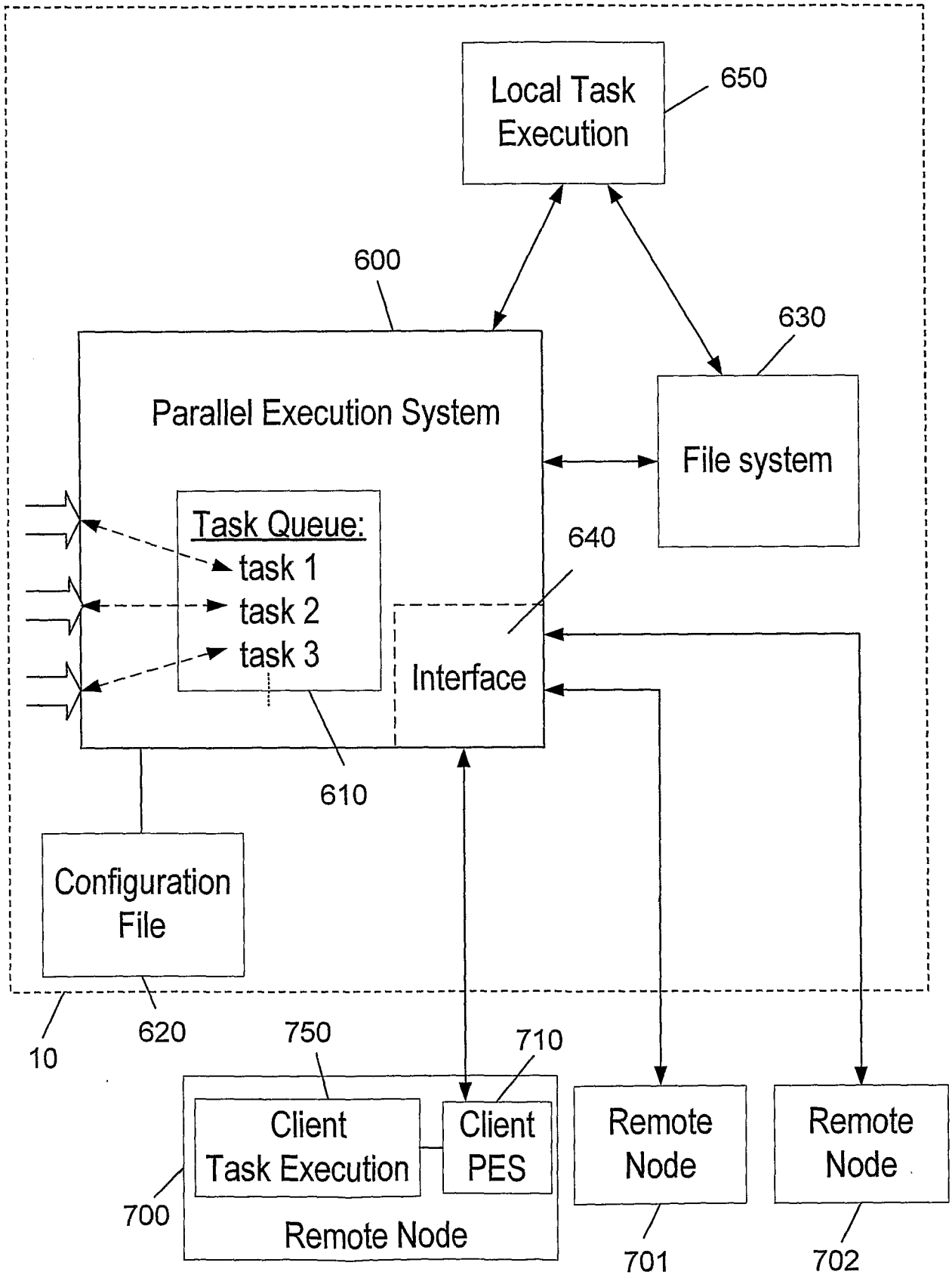

Local Task Execution
650
600
630
Parallel Execution System
File system
Task Queue:
task 1
task 2
task 3
640
Interface
610
Configuration File
620
10
750
710
Client Task Execution
Client PES
700
Remote Node
Remote Node
Remote Node
701
702

Fig. 3

INTERNATIONAL SEARCH REPORT

International application No
PCT/IL2007/000727

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F9/44

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, WPI Data, INSPEC

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	EP 0 859 314 A (SUN MICROSYSTEMS INC [US]) 19 August 1998 (1998-08-19) column 2, line 9 - line 27 column 4, line 20 - line 45 column 6, line 30 - line 41 column 7, line 3 - column 8, line 16 column 9, line 37 - line 55	1-9
X	US 2004/194060 A1 (OUSTERHOUT JOHN [US] ET AL OUSTERHOUT JOHN [US] ET AL) 30 September 2004 (2004-09-30) page 2, paragraph 18 page 3, paragraph 46 - page 4, paragraph 51	1-9
A	US 2004/194075 A1 (MOLCHANOV NIKOLAY [US] ET AL) 30 September 2004 (2004-09-30) the whole document	1-9

-/--

[X] Further documents are listed in the continuation of Box C.

[X] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
28 September 2007	08/10/2007

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Dewyn, Torkild

Form PCT/ISA/210 (second sheet) (April 2005)

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	FLECKENSTEIN C J ET AL: "USING A GLOBAL NAME SPACE FOR PARALLEL EXECUTION OF UNIX TOOLS" COMMUNICATIONS OF THE ASSOCIATION FOR COMPUTING MACHINERY, ACM, NEW YORK, NY, US, vol. 32, no. 9, 1 September 1989 (1989-09-01), pages 1085-1090, XP000070421 ISSN: 0001-0782 the whole document	1-9
A	US 2002/147855 A1 (LU YEN [CA]) 10 October 2002 (2002-10-10) the whole document	1-9
A	US 2005/268309 A1 (KRISHNASWAMY RAJA [US] ET AL) 1 December 2005 (2005-12-01) the whole document	1-9
A	US 2004/172626 A1 (JALAN ABHINAV [IN] ET AL) 2 September 2004 (2004-09-02) the whole document	1-9

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/IL2007/000727

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0859314 A	19-08-1998	NONE	
US 2004194060 A1	30-09-2004	EP 1623320 A2 WO 2004088510 A2	08-02-2006 14-10-2004
US 2004194075 A1	30-09-2004	WO 2004095271 A2	04-11-2004
US 2002147855 A1	10-10-2002	CA 2343437 A1	06-10-2002
US 2005268309 A1	01-12-2005	NONE	
US 2004172626 A1	02-09-2004	NONE	