US 20070234320A1

(19) United States

(12) Patent Application Publication (10) Pub. No.: US 2007/0234320 A1

Gu (43) Pub. Date: Oct. 4, 2007

(54) DISTRIBUTED PARALLEL BUILD SYSTEM

(75) Inventor: **Kevin Gu**, San Jose, CA (US)

Correspondence Address:
SCHWEGMAN, LUNDBERG, WOESSNER & KLUTH, P.A.
P.O. BOX 2938
MINNEAPOLIS, MN 55402

(73) Assignee: **eBay Inc.**

(21) Appl. No.: **11/526,310**

(22) Filed: **Sep. 25, 2006**

Related U.S. Application Data

(60) Provisional application No. 60/744,039, filed on Mar. 31, 2006.

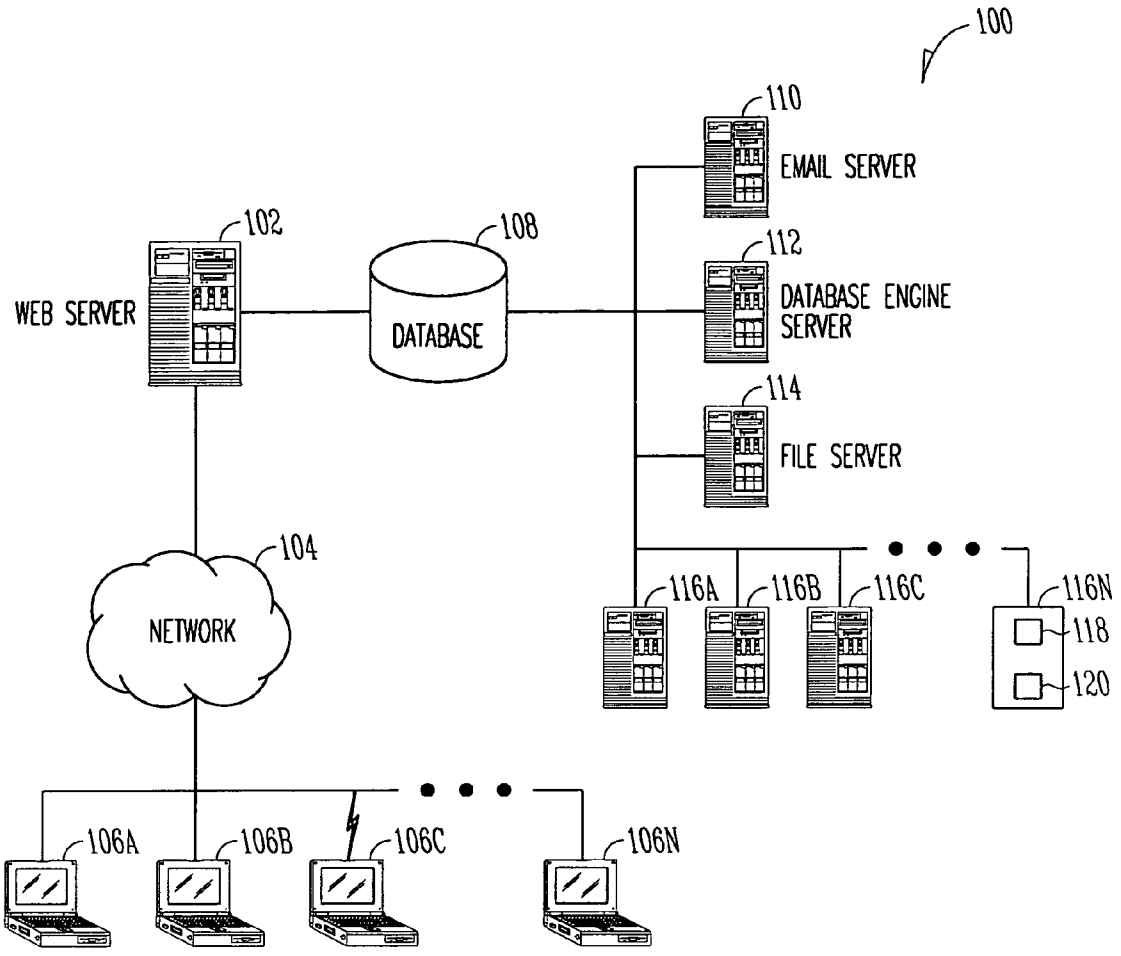
Publication Classification

(51) **Int. Cl.**
G06F 9/45 (2006.01)

(52) **U.S. Cl.** **717/140**

(57) **ABSTRACT**

This document describes, among other things, systems and methods for managing distributed parallel builds. A computer-implemented method to manage parallel builds, comprises identifying one or more software components in a software project, wherein each software component includes an executable binary file; determining a build configuration for each software component, wherein the build configuration includes a mapping from each software component to one or more build servers; and building each software component using the mapped one or more build servers in the corresponding build configuration, wherein the building includes compiling one or more source files associated with each software component to one or more object files, by distributing the one or more source files to one or more compilation machines.



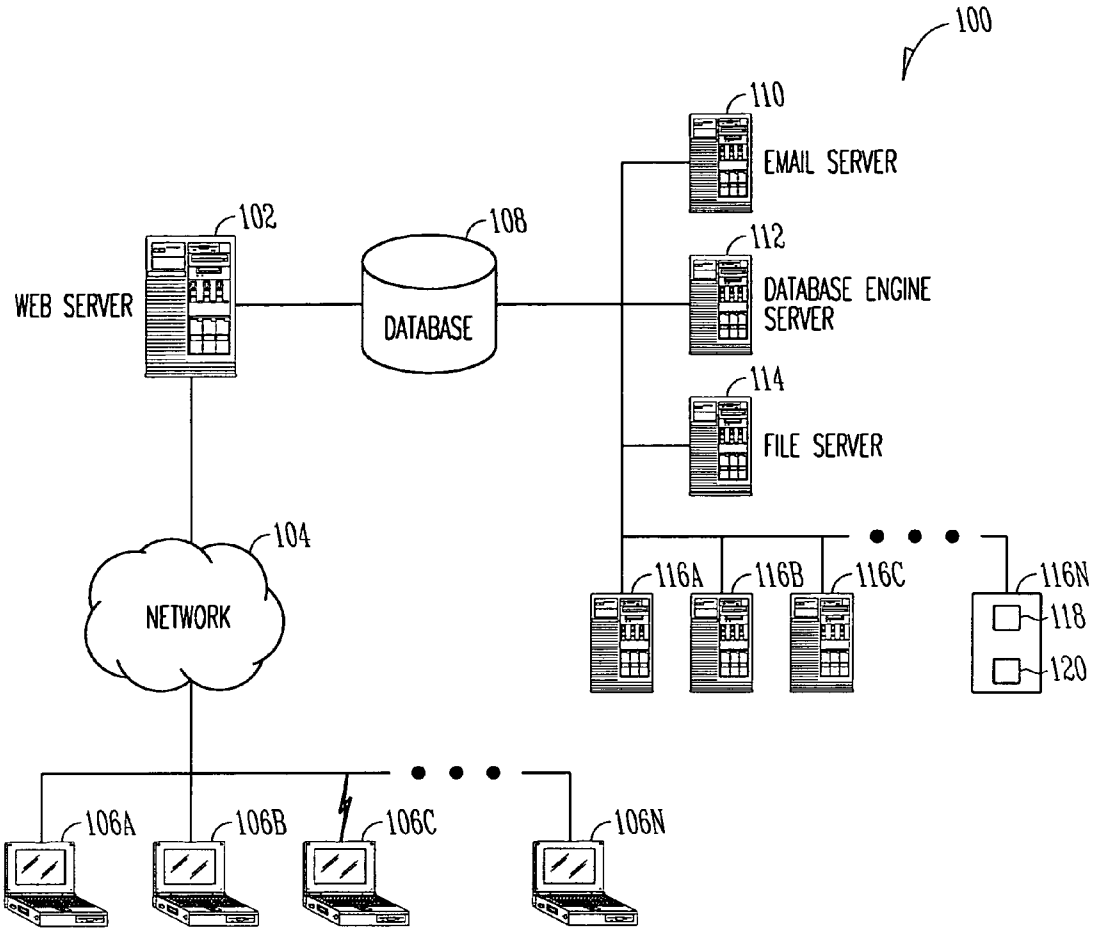

100
110
EMAIL SERVER
102
WEB SERVER
108
DATABASE
112
DATABASE ENGINE SERVER
114
FILE SERVER
104
NETWORK
116A
116B
116C
116N
118
120
106A
106B
106C
106N

FIG. 1

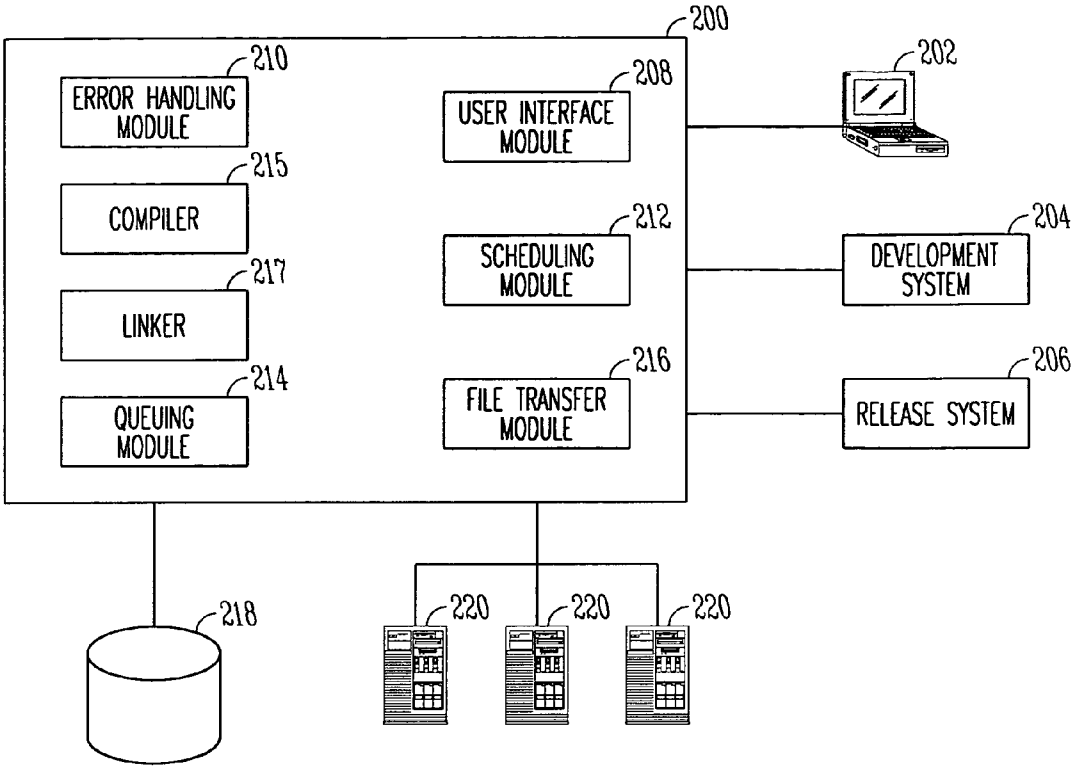

200
210
ERROR HANDLING MODULE
215
COMPILER
217
LINKER
214
QUEUING MODULE
208
USER INTERFACE MODULE
212
SCHEDULING MODULE
216
FILE TRANSFER MODULE
202
204
DEVELOPMENT SYSTEM
206
RELEASE SYSTEM
218
220
220
220

FIG. 2

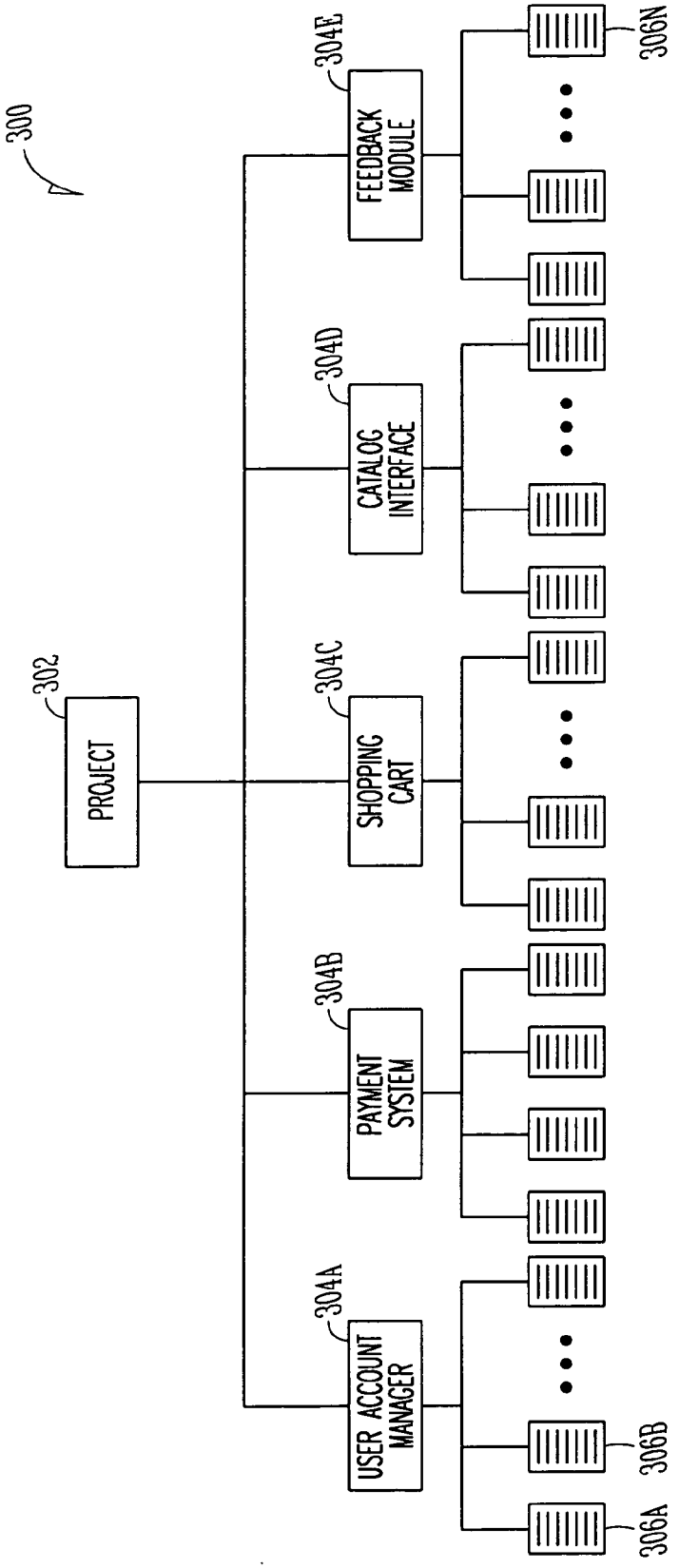

300
302
PROJECT
304A
USER ACCOUNT MANAGER
304B
PAYMENT SYSTEM
304C
SHOPPING CART
304D
CATALOG INTERFACE
304E
FEEDBACK MODULE
306A
306B
306N

FIG. 3

400

402 COMPONENT	404 SERVERS
USER ACCOUNT MODULE	DEV_0, DEV_1, DEV_2
PAYMENT SYSTEM	DEV_3, DEV_4
SHOPPING CART	DEV_5, DEV_7
CATALOG INTERFACE	DEV_10, DEV_8
FEEDBACK MODULE	DEV_11, DEV_12, DEV_6

FIG. 4

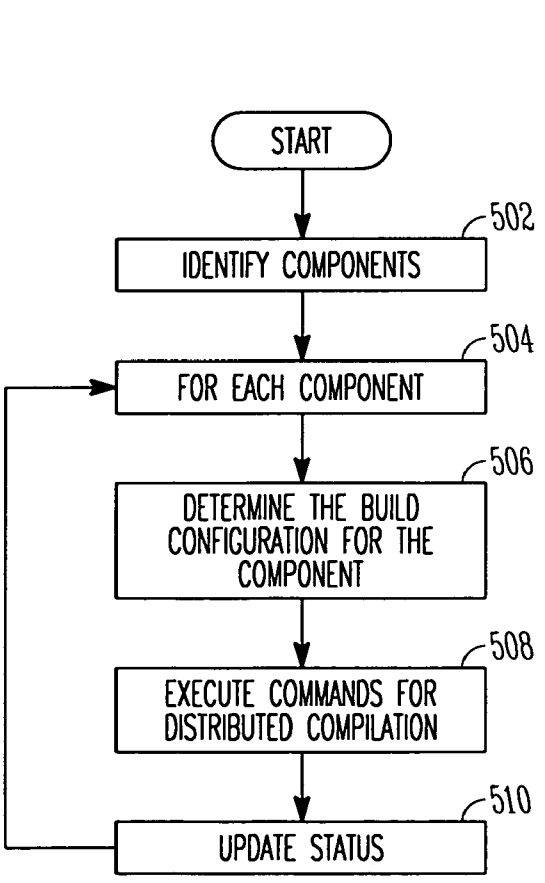


FIG. 5

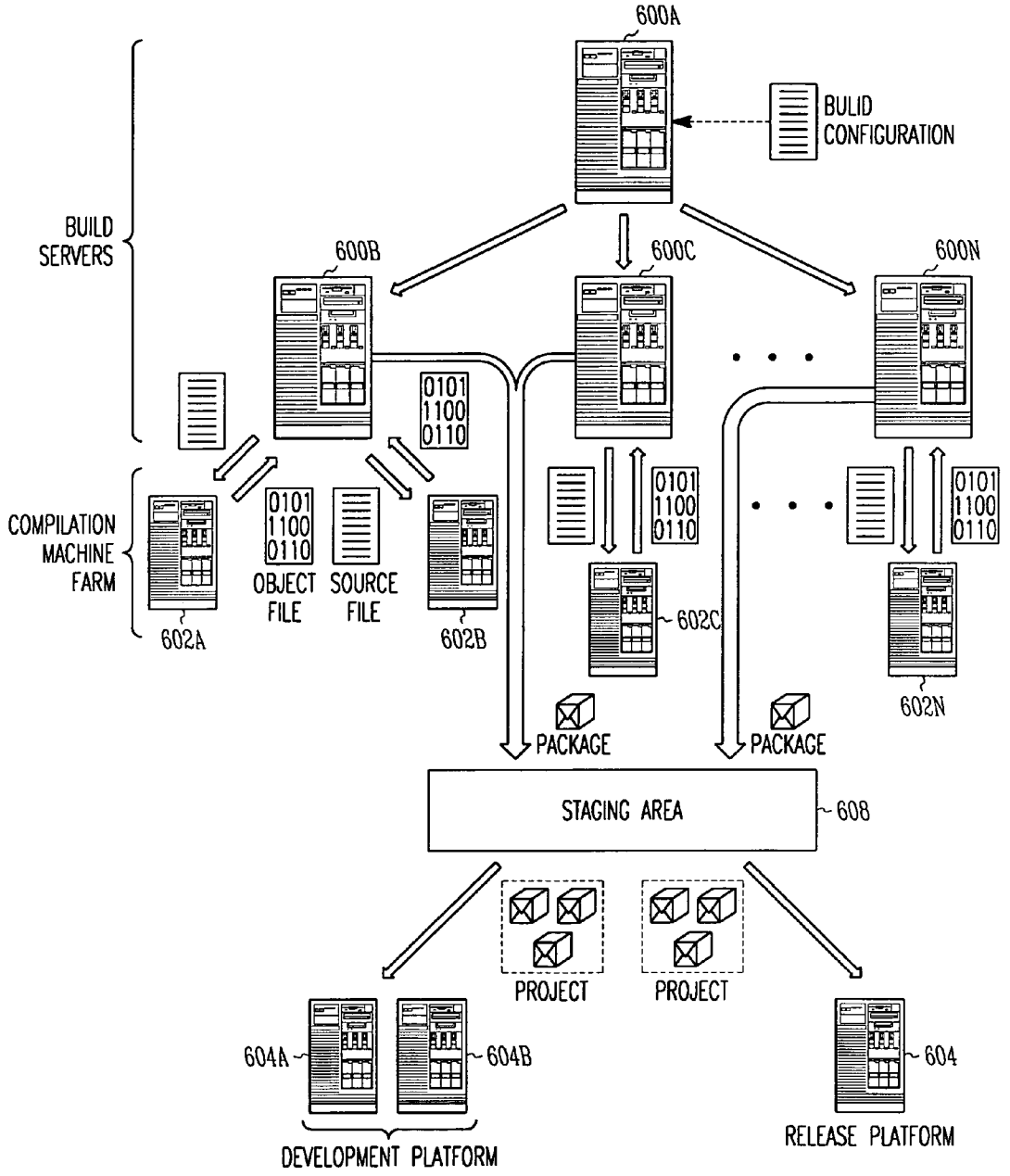

600A
BULID CONFIGURATION
BUILD SERVERS
600B
600C
600N
0101 1100 0110
COMPILATION MACHINE FARM
OBJECT FILE
SOURCE FILE
602A
602B
602C
602N
PACKAGE
PACKAGE
STAGING AREA
608
PROJECT
PROJECT
604A
604B
604
DEVELOPMENT PLATFORM
RELEASE PLATFORM

FIG. 6

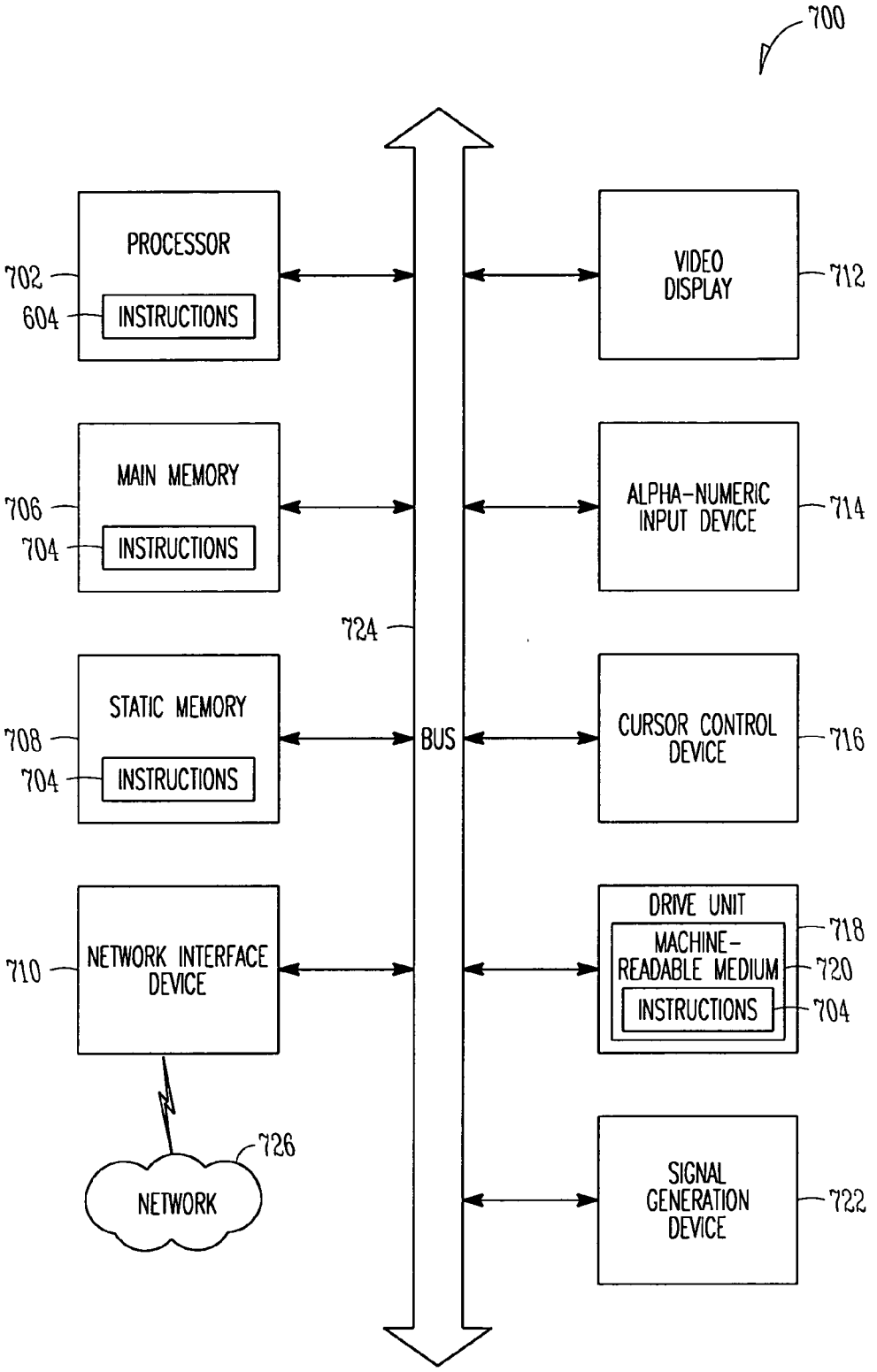

700
PROCESSOR
702
604
INSTRUCTIONS
MAIN MEMORY
706
704
INSTRUCTIONS
STATIC MEMORY
708
704
INSTRUCTIONS
NETWORK INTERFACE DEVICE
710
726
NETWORK
724
BUS
VIDEO DISPLAY
712
ALPHA-NUMERIC INPUT DEVICE
714
CURSOR CONTROL DEVICE
716
DRIVE UNIT
718
MACHINE-READABLE MEDIUM
720
INSTRUCTIONS
704
SIGNAL GENERATION DEVICE
722

FIG. 7

DISTRIBUTED PARALLEL BUILD SYSTEM

RELATED PATENT DOCUMENTS

[0001] This patent application claims the benefit of priority, under 35 U.S.C. Section 119(e), to U.S. Provisional Patent Application Ser. No. 70/744,039, entitled "Distributed Parallel Build System," filed on Mar. 31, 2006, the contents of which are hereby incorporated by reference in its entirety.

TECHNICAL FIELD

[0002] Embodiments relate generally to the field of software development, and more specifically to methods and systems that build software projects in parallel.

BACKGROUND

[0003] The software development process usually involves several steps including analyzing requirements, drafting specifications, designing the software architecture, coding, testing and debugging, and maintenance. During the coding, testing, and debugging stages some or all of a software project is built using tools such as a compiler and a linker. In a complex software project, builds may take long periods of time, causing an inefficient use of software development resources.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] FIG. **1** illustrates a schematic block diagram of a network-based system, in accordance with an example embodiment;

[0005] FIG. **2** illustrates a schematic block diagram of a build manager, in accordance with an example embodiment;

[0006] FIG. **3** illustrates a schematic block diagram of a component-based arrangement, in accordance with an example embodiment;

[0007] FIG. **4** is a chart illustrating build configurations for software components, in accordance with an example embodiment;

[0008] FIG. **5** illustrates a method for component-based distributed parallel builds, in accordance with an example embodiment;

[0009] FIG. **6** illustrates a schematic block diagram of build servers and compilation machines, in accordance with an example embodiment; and

[0010] FIG. **7** illustrates a diagrammatic representation of a machine in the exemplary form of a computer system, within which a set or sequence of instructions for causing the machine to perform any one of the methodologies discussed herein may be executed.

DETAILED DESCRIPTION

[0011] Methods and systems to manage software builds in a network-based system are described. In the following description, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of embodiments of the inventive subject matter. It will be evident, however, to one skilled in the art that embodiments of the inventive subject matter may be practiced without these specific details.

[0012] For the purposes of this document, "software component" includes any independent binary or independent executable software module, such as library files (e.g., dynamically linked libraries or DLL), executable files (e.g., applications or .exe files), or services (e.g., daemons). Other types of independent binaries are included as understood by one of ordinary skill in the art.

[0013] FIG. **1** illustrates a schematic block diagram of a network-based system **100**, in accordance with an example embodiment. The network-based system **100** includes a web server **102**, which can communicate over a network **104** with one or more terminals **106**A, **106**B, **106**C, . . . , **106**N. In various embodiments, portions of the network **104** may include wired or wireless networking. The terminals **106** can connect to the network **104** using wired or wireless communication. The web server **102** is communicatively coupled to a database **108** and other backend servers, such as an email server **110**, a database engine server **112**, and a file server **114**. Additionally, the system **100** includes one or more build servers **116**A, **116**B, **116**C, . . . , **116**N. In embodiments, the build servers **116** may include any type of computer including a laptop, desktop, blade server, network server, or the like. In addition, build servers **116** may include one or more software compilers **118** or linkers **120**.

[0014] In an embodiment, a user (e.g., a software developer) can use a terminal machine **106** to control software builds using a web-based user-interface provided by the web server **102**. In an embodiment, a user (e.g., a software developer) can use a terminal machine **106** to control software builds using a web-based user-interface provided by the web server **102**. During a typical software development phase, the user may write and edit files, which are part of a software project. At some time, the user may desire to build the project. Building the project may involve compiling one or more files and then linking them into one or more files (e.g., executable or library files). In an embodiment, the user can initiate such a build process using the user-interface provided by the web server **102**. Upon receiving such a request, the web server **102** communicates with the database engine server **112**. Communication may include information such as user identification, project identification, build destination, and other pertinent information. The database engine server **112** can then communicate with the database **108** to determine the files needed and the correct commands to issue to the build servers **116**. Files may be stored in the database **108** or on a file server **114**. Files for each particular software component are associated with one or more build servers **116** in a distributed manner. In an embodiment, each build server **116** contains a compiler and a linker. The files are transmitted to the associated build servers **116**, where they are compiled and linked. After each software component is compiled and linked on the distributed build servers **116**, the software project (e.g., the collection of the particular software components) is transmitted to a development or a release area.

[0015] FIG. **2** illustrates a schematic block diagram of a build manager **200**, in accordance with an example embodiment. The build manager **200** may operate, in some embodiments, as a build server **220**. In such an embodiment, the build manager **200** can compile one or more files while also coordinating the distributed parallel build process. Alternatively, the build manager **200** may be solely operated as a controller machine: taking commands from a client machine **202**, managing the build process, and placing final builds on either a development system **204** or a release system **206**.

[0016] In an embodiment, the build manager **200** includes a user interface module **208**, an error handling module **210**,

a scheduling module **212**, a queuing module **214**, a file transfer module **216**, a compiler **215**, and a linker **217**. Users (e.g., software developers) may connect with the build machine **200** from their client machines **202** to issue one or more commands that control software builds. For example, a user may issue a command via the user interface module **208** to schedule a build or a series of builds, cancel a build or check the status of a build.

[0017] The scheduling module **212** may be used to schedule one or more builds in the future. In an embodiment, users can schedule one or more builds to commence at a specific time or date. In another embodiment, users can schedule periodic or recurring builds. Schedules of periodic or recurring builds may have a terminating date or time, such that software projects will regularly re-build until the terminating date or time.

[0018] The error handling module **210** detects errors that may occur before or during a build. In an embodiment, detected errors are logged to a database **218** or a file. Users may, in some embodiments, view the errors stored in the database **218** using the user interface module **208**. In some embodiments, the error handling module **210** may communicate with one or more modules to control current or later builds. For example, if a build fails and the error handling module **210** detects a certain type of error or a certain degree of error, the error handling module **210** may communicate with the scheduling module **212** to discontinue future builds or defer the next scheduled build until, for example, the user is notified and issues a command to continue the scheduled builds. In another embodiment, after detecting an error, the error handling module **210** may communicate with the queuing module **214** to remove any queued portions of the build not yet in progress, thereby terminating the build.

[0019] The queuing module **214** manages user build requests using one or more queues, in embodiments. In an embodiment, queuing is sorted by a first-come, first-served priority basis. In other embodiments, queuing is prioritized using factors such as a project type, a requestor, a project size, a project priority, or the like. In an embodiment, the queuing module **214** may communicate with the scheduling module **212** to receive scheduled builds. The queuing module **214** may then insert the scheduled builds into a queue based on one or more of the factors noted above. In an embodiment, queuing is performed at the project level, such that a project may not begin building until a previous project is completed. In another embodiment, queuing is performed at a more granular level, such as at a component or file level. It may be advantageous to manage queues at a finer granularity to reduce build server **220** idle time. In an embodiment, the queuing module **214** communicates with the database **218** to update the build status. For example, the queuing module **214** may communicate with the user interface module **208** to provide an indication of which software project or software component is currently being built, which project or component is next in the queue, or indications of success or error conditions existing in the current software build.

[0020] In an embodiment, the file transfer module **216** can transfer source files to one or more build servers **220**. Source files may be stored on a file server, removable storage media, or a structured storage system, such as a database, version control system, or the like. In one embodiment, source files are stored in Rational ClearCase provided by IBM, Inc. In an embodiment, the build servers **220** compile the source files to object files. In another embodiment, the build servers **220** manage the distributed compilation of the source files across one or more compilation machines. The file transfer module **216** may then transfer the resultant object file from each build server **220** to another server, which may in some embodiments be the build manager **200**, where linking is performed. In an alternative embodiment, linking is performed by one or more build servers **220** and the file transfer module **216** accesses the linked executable file. In either embodiment, the linked executable file is eventually transferred to the development system **204** or the release system **206**. In embodiments, file transfers are performed using secured (e.g., Secure Shell (SSH)) or unsecured (e.g., TCP/IP) protocols. In an embodiment, the file transfer module **216** communicates with the database **218** to update the status of file transfers.

[0021] FIG. **3** illustrates a schematic block diagram of a component-based arrangement **300**, in accordance with an example embodiment. In an embodiment, a project **302** may be divided into several software components. In one example embodiment, the project may include an online commerce system, where the software components include a user account manager **304**A, a payment system **304**B, a shopping cart **304**C, a catalog interface **304**D, and a feedback module **304**E. Each component **304** may be logically divided further into separate files or sets of files **306**A, **306**B, . . . , **306**N representing a subdivision of the component **304** related in some manner, such as by related functionality, complexity, or the like. In an embodiment, the subdivided files or sets of files **306** are organized in a manner that increases the speed or efficiency of the build process. For example, one or more complex files may be associated with a more powerful build server **116** (FIG. **1**), whereas less complex files may be associated with a less powerful build server **116**, in order to seek overall efficiency gains.

[0022] FIG. **4** is a chart **400** illustrating build configurations for software components, in accordance with an example embodiment. As depicted in FIG. **4**, a build configuration for a software component includes a mapping from a particular software component to one or more servers. For example, each software component in the component column **402** is associated with one or more computing machines in the server column **404**. A command can be issued to compile a component in a distributed manner using the machines identified in the server column **404**. In an embodiment, the machines, as described in column **404**, include a build server. In another embodiment, the machines include a computing device dedicated to compiling or assembling source code to object code, such as a compilation machine. In an embodiment, a software tool is used to manage and facilitate distributed compilation, such as for example, distcc by Martin Pool. The number of servers and which servers are used for each component may be determined by one or more factors, including the size of the component, the complexity of the component, the computational capability of each build server, the typical frequency of changes to a particular component, the network capabilities to one or more servers, and other factors that may be considered when attempting to optimize build times. In an embodiment, where two or more components are of sufficiently simple complexity to maintain overall build efficiency, portions or all of the components may be compiled or built on a single build server.

[0023] In an embodiment, mapping is performed by the build manager **200** on a first-come first-served basis. For example, a list of one or more software components may be submitted to a user interface provided by the user interface module **208**. The build manager **200** may process the list of software components from first to last and map each software component to one or more build servers depending on one or more factors, as previously described. The build manager **200** may be aware of how many build servers are available and what each build server's configuration is, for example memory, processing capability, or the like. Using such information, the build manager **200** may adaptively map (assign) software components to build servers in an effort to balance processing duties or an attempt to achieve an overall maximized operating efficiency. In another embodiment, mapping is performed based on one or more characteristics of the software components, such as the size, complexity, age, name, priority, language, or the like. Using characteristics of the software components may be advantageous by providing the best resources to the most complex components before mapping other less complex components, which may not fully maximize a build server's capabilities. In another embodiment, mapping is performed by dividing the number of available build servers by the number of software components and then assigning the allotted number of build servers to each software component.

[0024] FIG. **5** illustrates a method **500** for component-based distributed parallel builds, in accordance with an example embodiment. At **502**, each component is identified. In one embodiment, a command is issued by a user of the system **100** using a user-interface provided on a terminal machine **106**. The command can include an indication of one or more components to build. The method **500** can parse the command to determine the components.

[0025] For each component **504**, the method **500** determines the build configuration for the component **506**. For example, using a table similar to the one illustrated in FIG. **4**, the method **500** can determine which build servers **116** (FIG. **1**) will be targeted for the distributed compilation of the particular component.

[0026] At **508**, one or more commands are issued to distributively compile the component. In an embodiment, the command identifies one or more build servers **116** to be used to compile the particular component. In an embodiment, one or more components are built using one or more assigned build servers **116**, where building a component includes compiling and linking the component's source files. In another embodiment, component source files are only compiled on build servers **116** in a distributed manner, and linking the resulting object code is performed on a different computer. In another embodiment, build servers **116** control the distributed compilation of one or more components using one or more compilation machines.

[0027] At **510**, the current build status is updated for later use, for example, by a report or a user-interface screen to be provided to a user showing the current status of each component build.

[0028] In certain embodiments, some or all of the builds can be scheduled. In other embodiments, some or all of the builds can be performed in serial, parallel, or a combination. This may be necessary, for example, because of inherent dependencies between different units of software.

[0029] In an embodiment, a build is distributed over two or more CPUs in a single machine, such that, for example, each component is assigned and associated with one or more CPUs in the machine. In another embodiment, a machine has a multiple-core CPU (e.g., AMD Athlon X2 series and Intel Pentium D processors) and a build can be distributed over two or more CPU cores in a single machine. In a further embodiment, a machine may have multiple CPUs, each with multiple CPU cores, and systems and methods described herein can adaptively associate and assign components in a project to particular CPUs or CPU cores or any combination of the two. In a further embodiment, builds can be distributed over several multi-processor machines, where each machine's processors may or may not have multiple cores. Components can then be assigned to a particular processor on a particular machine or even with more granularities, such as by assigning a certain component build to one or more processor cores on a particular machine or across several machines.

[0030] FIG. **6** illustrates a schematic block diagram of build servers **600** and compilation machines **602**, in accordance with an example embodiment. In one configuration, multiple build servers **600**A, **600**B, **600**C, . . . , **600**N are arranged in a hierarchal tree, such that a root build server **600**A has the task of distributing one or more software components of a software project to one or more component-level build servers **600**B, **600**C, . . . , **600**N. In some embodiments, the root build server **600**A includes a linker, a packager (e.g., Red Hat Package Manager) and other software to distribute and manage distributed compilations such that the root build server **600**A may act as a component-level build server during a separate project build. In an embodiment, the packager includes software to create a software package using a particular file format, such that a management tool can install, update, uninstall, verify and query software packaged in the format. Component-level build servers **600**B, **600**C, . . . , **600**N include a linker, software to distribute and manage distributed compilations, and a packager. Compilation machines **602**A, **602**B, **602**C, . . . , **602**N includes one or more compilers and are configured as dedicated machines with a primary task of compiling source code to object code, in an embodiment.

[0031] In an embodiment, after a build command is submitted to the root build server **600**A, components of the software project are distributed to the component-level build servers **600**B, **600**C, . . . , **600**N. In another embodiment, each component-level build server **600**B, **600**C, . . . , **600**N is provided with a component identifier, such as a component name, and may retrieve the source files associated with the component from a central repository, such as a version control system. At each component-level build server **600**B, **600**C, . . . , **600**N, the source files that are associated with the component are distributed across the compilation machines **602** using software, such as distcc, where the source files are compiled to object files. In one embodiment, a configuration file maps software components to one or more compilation machines **602**. The configuration file may be stored in a central repository. When a component-level build server **600**B, **600**C, . . . , **600**N receives a build command identifying the component to build, the associated configuration file may be retrieved and used to determine the target compilation machines **602** for the distributed compilation. After the source files are compiled, the object files are linked at the component-level build server **600**B, **600**C, . . . , **600**N

and the resulting software component (e.g., executable file or library file) is packaged using the packager. The package can then be transferred to a development platform **604** or a release platform **606**, where it can be installed and used or tested. In an embodiment, development platform **604** may include one or more development servers **604**A, **604**B to provide parallel development and testing. In an embodiment, packages from component-level build server **600**B, **600**C, . . . , **600**N are mustered at a staging area **608**, before being distributed to the development platform **604** or the release **606**. In various embodiments, the staging area may include one or more file servers, database servers, or the like, to temporarily store the packages before distribution. The staging area **608** may validate that all packages in a software project are present before transferring the project (e.g., in the form of a complete set of packages) to the appropriate target platform.

[0032] FIG. **7** illustrates a diagrammatic representation of a machine in the exemplary form of a computer system **700** within which a set or sequence of instructions, for causing the machine to perform any one of the methodologies discussed herein, may be executed. In alternative embodiments, the machine may comprise a computer, a network router, a network switch, a network bridge, Personal Digital Assistant (PDA), a cellular telephone, a web appliance, set-top box (STB) or any machine capable of executing a sequence of instructions that specify actions to be taken by that machine.

[0033] The computer system **700** includes a processor **702**, a main memory **706** and a static memory **708**, which communicate with each other via a bus **724**. The computer system **700** may further include a video display unit **712** (e.g., a liquid crystal display (LCD) or a cathode ray tube (CRT)). The computer system **700** also includes an alphanumeric input device **714** (e.g., a keyboard), a cursor control device **716** (e. g., a mouse), a disk drive unit **717**, a signal generation device **722** (e.g., a speaker) and a network interface device **710** to interface the computer system to a network **726**.

[0034] The disk drive unit **718** includes a machine-readable medium **720** on which is stored a set of instructions or software **704** embodying any one, or all, of the methodologies described herein. The software **704** is also shown to reside, completely or at least partially, within the main memory **706** and/or within the processor **702**. The software **704** may further be transmitted or received via the network interface device **710**. For the purposes of this specification, the term "machine-readable medium" shall be taken to include any medium which is capable of storing or encoding a sequence of instructions for execution by the machine and that cause the machine to perform any one of the methodologies of the inventive subject matter. The term "machine-readable medium" shall accordingly be taken to include, but not be limited to, solid-state memories, optical and magnetic disks, and carrier wave signals. Further, while the software is shown in FIG. **7** to reside within a single device, it will be appreciated that the software could be distributed across multiple machines or storage media, which may include the machine-readable medium.

[0035] The foregoing description of specific embodiments reveals the general nature of the inventive subject matter sufficiently that others can, by applying current knowledge, readily modify and/or adapt it for various applications without departing from the generic concept. Therefore, such adaptations and modifications are within the meaning and range of equivalents of the disclosed embodiments. The phraseology or terminology employed herein is for the purpose of description and not of limitation. Accordingly, the inventive subject matter embraces all such alternatives, modifications, equivalents and variations as fall within the spirit and broad scope of the appended claims.

[0036] Method embodiments described herein may be computer-implemented. Some embodiments may include computer-readable media encoded with a computer program (e.g., software), which includes instructions operable to cause an electronic device to perform methods of various embodiments. A software implementation (or computer-implemented method) may include microcode, assembly language code, or a higher-level language code, which further may include computer readable instructions for performing various methods. The code may form portions of computer program products. Further, the code may be tangibly stored on one or more volatile or non-volatile computer-readable media during execution or at other times. These computer-readable media may include, but are not limited to, hard disks, removable magnetic disks, removable optical disks (e.g., compact disks and digital video disks), magnetic cassettes, memory cards or sticks, random access memories (RAMS), read only memories (ROMs), and the like.

[0037] In the foregoing description of various embodiments, reference is made to the accompanying drawings, which form a part hereof and show, by way of illustration, specific embodiments in which the inventive subject matter may be practiced. Various embodiments are described in sufficient detail to enable those skilled in the art to practice the inventive subject matter, and it is to be understood that other embodiments may be utilized, and that process or mechanical changes may be made, without departing from the scope of the inventive subject matter.

[0038] Embodiments of the inventive subject matter may be referred to, individually and/or collectively, herein by the term "inventive subject matter" merely for convenience and without intending to voluntarily limit the scope of this application to any single inventive subject matter or inventive concept if more than one is, in fact, disclosed. It will be recognized that the methods of various embodiments can be combined in practice, either concurrently or in succession. Various permutations and combinations may be readily apparent to those skilled in the art.

1. A computer-implemented method to manage parallel builds, comprising:

identifying one or more software components in a software project, wherein each software component includes an executable binary file;

determining a build configuration for each software component, wherein the build configuration includes a mapping from each software component to one or more build servers; and

building each software component using the mapped one or more build servers in the corresponding build configuration, wherein the building includes compiling one or more source files associated with each software component to one or more object files, by distributing the one or more source files to one or more compilation machines.

2. The method of claim **1**, further comprising scheduling one or more builds.

3. The method of claim 2, wherein the scheduling is periodic or recurring.

4. The method of claim 1, further comprising queuing one or more builds.

5. The method of claim 4, wherein the queuing is based on one or more of a project type, a requestor, a project size, a project priority or a project request time.

6. The method of claim 4, wherein the queuing is organized by one or more of a project-level, a component-level, or a file-level.

7. The method of claim 1, wherein building each software component includes:

linking the one or more object files associated with each software component;

packaging each software component; and

transferring each packaged software component to one of a development area or a release area.

8. A device comprising:

a user-interface module;

a queuing module, coupled to the user-interface module, the queuing module configured to maintain a build queue, the build queue including one or more build requests, each build request including one or more software components; and

a file transfer module, coupled to the queuing module, configured to transfer one or more files associated with one or more software components associated with a build request in the build queue to one or more build servers according to a build configuration, wherein the build configuration includes a mapping from each software component to one or more build servers.

9. The device of claim 8, further comprising a scheduling module, coupled to the user-interface module and the queuing module, configured to manage one or more scheduled builds.

10. The device of claim 8, further comprising an error handling module, coupled to the user-interface module and the queuing module, configured to detect one or more error conditions and perform one or more actions in response to the detected error conditions.

11. The device of claim 10, wherein the actions include one or more of reporting the error condition to the user-interface, logging the error condition, or ignoring the error condition.

12. A computer-assisted method of managing parallel builds, comprising:

means for organizing a software project into one or more software components, wherein each software component is associated with an executable binary file;

means for determining a build configuration for each software component, wherein the build configuration includes a mapping from each software component to one or more build servers; and

means for compiling each software component using the build configuration.

13. A computer-readable medium including instructions that, when performed by a computer, cause the computer to:

identify one or more software components in a software project, wherein each software component includes an executable binary file;

determine a build configuration for each software component, wherein the build configuration includes a mapping from each software component to one or more build servers; and

compile each software component using the mapped one or more build servers in the corresponding build configuration.

14. The computer-readable medium of claim 13, further comprising instructions that cause the computer to schedule one or more builds.

15. The computer-readable medium of claim 13, further comprising instructions that cause the computer to queue one or more builds.

16. The computer-readable medium of claim 15, wherein the queuing is based on one or more of a project type, a requester, a project size, a project priority or a project request time.

17. The computer-readable medium of claim 15, wherein the queuing is organized by one or more of a project-level, a component-level, or a file-level.

* * * * *