July 2022

EFFICIENT SEARCHING DURING APPLICATION LAUNCH PROCESS

Jason Chihhao Lee

Alex Lin

Jabez Chung

Paservan Yu

James C

See next page for additional authors

Inventor(s)
Jason Chihhao Lee, Alex Lin, Jabez Chung, Paservan Yu, James C, and Peggy Kao

EFFICIENT SEARCHING DURING APPLICATION LAUNCH PROCESS

ABSTRACT

An application (e.g., a camera application, an entertainment application, a productivity application, a communication application, etc.) of a computing device (e.g., a smartphone, a laptop computer, a tablet computer, a smartwatch, etc.) may use a search module to efficiently search for application content (e.g., a graphical interface, a frame, etc.) to be displayed when the computing device launches the application. In some examples, the search module may be a Fibonacci-based algorithm that searches an application content repository in a sequence based on the Fibonacci sequence (e.g., 1, 2, 3, 5, 8 , etc.). In other examples, the search module may be a binary search algorithm that searches the application content repository in a sequence based on powers of two. Experiments show that both the Fibonacci-based algorithm and the binary search algorithm can find launch application content faster than some existing solutions. Thus, usage of search module 112 may advantageously decrease time-to-initial-display (TTID).

DESCRIPTION

FIG. 1 below is a conceptual diagram illustrating a computing device 100 that executes a search module in accordance with techniques of this disclosure. As shown in FIG. 1, computing device 100 includes one or more displays 102 (“display 102”), one or more processors 104, and one or more storage devices 106. As further shown in FIG. 1, storage devices 106 stores one or more applications 108 (“application 108”), an application content repository 110, and a search module 112.

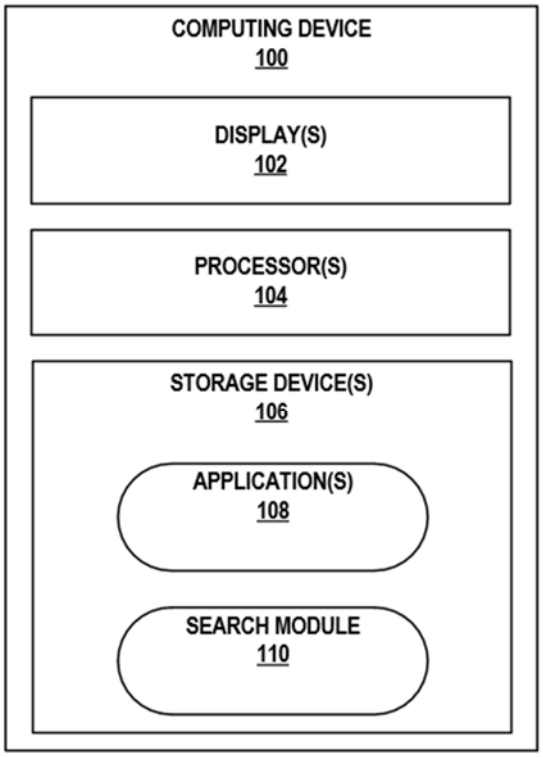

FIG. 1

In the example of FIG. 1, computing device 100 represents an individual mobile or non-mobile computing device. Examples of computing device 100 include a mobile phone, a tablet computer, a laptop computer, a desktop computer, a set-top box, a television, a wearable device (e.g., a computerized watch, computerized eyewear, computerized headphones, computerized gloves, etc.), a gaming system, a media player, an e-book reader, a mobile television platform, an automobile navigation or infotainment system, etc.

Display 102 of computing device 100 may be a display that functions as an output device. For example, display 102 may function as an output (e.g., display) device using any of one or more display components, such as a liquid crystal display (LCD), dot matrix display, light emitting diode (LED) display, microLED display, organic light-emitting diode (OLED) display,

e-ink, active-matrix organic light-emitting diode (AMOLED) display, or similar monochrome or color display capable of outputting visible information to a user of computing device 100.

Processors 104 may implement functionality and/or execute instructions associated with computing device 100. Examples of processors 104 include one or more of an application specific integrated circuit (ASIC), a field programmable gate array (FPGA), an application processor, a display controller, an auxiliary processor, a central processing unit (CPU), a graphics processing unit (GPU), one or more sensor hubs, and any other hardware configure to function as a processor, a processing unit, or a processing device. In some examples, processors 104 may represent a system on a chip (SoC) that includes an integrated circuit for implementing one or more of the above referenced examples of processors 104, along with supporting memory and/or storage, and possibly various interfaces, modems, etc. as a single package.

Storage devices 106 may include one or more computer-readable storage media. For example, storage devices 106 may be configured for long-term, as well as short-term storage of information, such as instructions, data, or other information used by computing device 100. In some examples, storage devices 106 may include non-volatile storage elements. Examples of such non-volatile storage elements include magnetic hard disks, optical discs, solid state discs, and/or the like. In other examples, in place of, or in addition to the non-volatile storage elements, storage devices 106 may include one or more so-called “temporary” memory devices, meaning that a primary purpose of these devices may not be long-term data storage. For example, the devices may comprise volatile memory devices, meaning that the devices may not maintain stored contents when the devices are not receiving power. Examples of volatile memory devices include random-access memories (RAM), dynamic random-access memories (DRAM), static random-access memories (SRAM), etc.

When computing device 100 launches (e.g., starts, executes, etc.) application 108, application 108 may perform an application launch process. During the application launch process, display 102 may initially display a default graphical interface or frame, such as a blank screen. Also during the application launch process, application 108 may search application content repository 110 for the application content (e.g., a specific frame) to be displayed when computing device 100 launches application 108 ("launch application content"). For example, application 108 may search an index of frames stored within application content repository 110 for the launch application content. Responsive to finding the launch application content, applications 108 may cause display 102 to stop displaying the default GUI and instead display a GUI representing the launch application content ("launch GUI").

The period of time beginning with the launching of application 108 and ending with the display of the launch GUI is referred to here as time-to-initial-display (TTID). TTID may be correlated with the size of application content repository 110. For example, if application content repository 110 is large, application 108 may require more time to find the application launch content, making TTID longer. A user of computing device 100 may find a long TTID bothersome or annoying. Accordingly, decreasing the time required for applications 108 to find the launch application content may be desirable.

In accordance with techniques of this disclosure, applications 108 may use search module 112 to efficiently search application content repository 110. Search module 112 may search ranges of the index of application content, where each range include a portion of the application content stored within application content repository 110. In some examples, search module 112 may be a Fibonacci-based algorithm that searches application content repository 110 in a sequence based on the Fibonacci sequence (e.g., 1, 1, 2, 3, 5, 8, etc.) and/or a modified Fibonacci

sequence (e.g., 1, 2, 3, 5, 8, etc.). Per the Fibonacci-based algorithm, search module 112 may define the sizes of the ranges based on the Fibonacci sequence multiplied by 100. In other examples, search module 112 may be a binary search algorithm that searches application content repository 110 in a sequence based on powers of two. Per the binary search algorithm, search module 112 may define the sizes of the ranges by iteratively dividing the index of application content in half. Experiments show that both the Fibonacci-based algorithm and the binary search algorithm can find launch application content in application content repository 110 faster than some existing solutions. Thus, usage of search module 112 may advantageously decrease TTID.

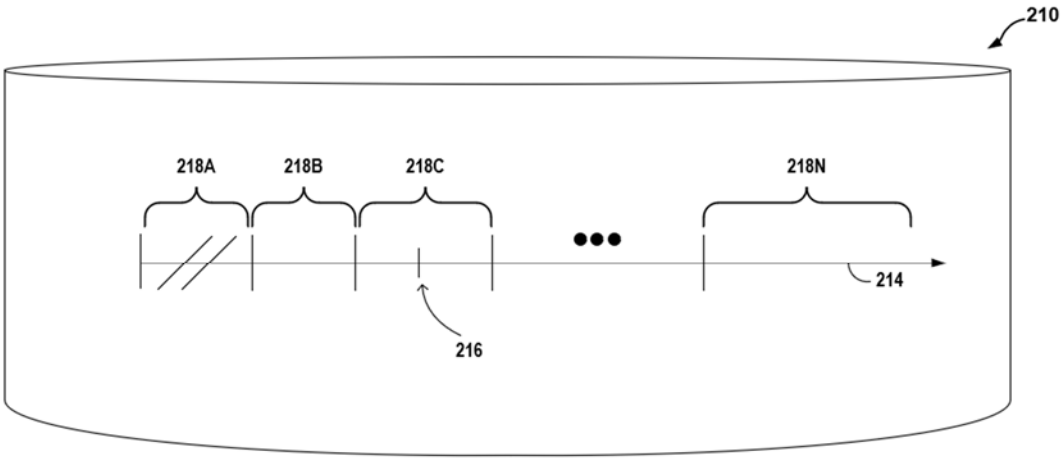

FIG. 2

FIG. 2 is a conceptual diagram illustrating an example operation of search module 112. As shown in FIG. 2, search module 112 may search an application content repository 210 for launch application content. For example, during the application launch process, search module 112 may search an index 214 of application content (e.g., frames) stored within application content repository 210 for an application content 216 to be displayed when computing device 100 launches application 108 (“launch application content 216”). Index 214 may include ranges,

such as ranges 218A-218N (collectively, “ranges 218”) that each include a portion of the application content stored within application content repository 210.

Search module 112 may search one or more ranges 218 for application launch content 216. In some examples, search module 112 may be a Fibonacci-based algorithm. Per the Fibonacci-based algorithm, search module 112 may define the sizes of ranges 218 based on the Fibonacci sequence multiplied by 100. For example, range 218A may include 100 frames of index 214, range 218B may include 200 frames of index 214, range 218C may include 300 frames of index 214, range 218D may include 500 frames of index 214, range 218E may include 800 frames of index 214, etc. Accordingly, range 218A may include frames 1-100 of index 214, range 218B may include frames 101-300 of index 214, range 218C may include frames 301-600 of index 214, range 218D may include frames 601-1100 of index 214, range 218E may include frames 1101-1900 of index 214, etc.

Search module 112 may iteratively search ranges 218 for launch application content 216 in the order of smallest range to the largest range. For instance, in the above example, search module 112 may search range 218A first, range 218B second, range 218C third, range 218D fourth, range 218E fifth, etc. Search module 112 may search ranges 218 in a forward direction (e.g., from the lower bound of a range to the upper bound of the range). For example, search module 112 may search range 218B from frame 101 to 300 for application launch content 216. Alternatively, search module 112 may search ranges in a reverse direction (e.g., from the upper bound of a range to the lower bound of the range). For example, search module 112 may search range 218B from frame 300 to 101 for application launch content 216.

When search module 112 finds application launch content 216 within one of ranges 218, search module 112 may stop searching. For example, application launch content may be frame

361 in index 214. In the first search iteration, search module 112 may search range 218A, which may include frames 1-100, in the reverse direction. In the second search iteration, search module 112 may search range 218B, which may include frames 101-300, in the reverse direction. In the third search iteration, search module 112 may search range 218C, which may include frames 301-600, in the reverse direction. Because frame 361 is within frames 301-600 and thus within range 218C, search module 112 may find application launch content 216 and stop searching. As a result, search module 112 may not search the remaining ranges of index 214 (e.g., 218D-218N).

In some examples, it may be assumed (e.g., because of industry practice) that application launch content 216 is not within an initial range of index 214 (e.g., frames 1-100 of index 214). In these examples, search module 112 may not search the initial range of index 214 for application launch content 216. The initial range of index 214 may correspond to range 218A. Accordingly, search module 112 may not search range 218A. Instead, search module 112 may search range 218B first, range 218C second, etc., for application launch content 216.

In some examples, search module 112 may be a binary search algorithm that is based on powers of two. Per the binary search algorithm, search module 112 may define the sizes of ranges 218 by iteratively dividing index 214 in half. For example, if index 214 includes 800 frames, range 218A may include 400 frames of index 214, range 218B may include 200 frames of index 214, range 218C may include 100 frames of index 214, range 218D may include 50 frames of index 214, range 218E may include 25 frames of index 214, etc. Search module 112 may round the sizes of ranges 218 so that each of ranges 218 includes an integer number of frames.

Search module 112 may iteratively search ranges 218 for launch application content 216 in the order of largest range to the smallest range. For instance, in the above example, search module 112 may search range 218A first, range 218B second, range 218C third, range 218D fourth, range 218E fifth, etc. Search module 112 may search ranges 218 in a forward direction. Alternatively, search module 112 may search ranges in a reverse direction.

When search module 112 finds application launch content 216 within one of ranges 218, search module 112 may stop searching. For example, application launch content may be frame 689 in index 214, which includes 800 frames. In the first search iteration, search module 112 may search range 218A, which may include frames 1-400, in the reverse direction. In the second search iteration, search module 112 may search range 218B, which may include frames 401-600, in the reverse direction. In the third search iteration, search module 112 may search range 218C, which may include frames 601-700, in the reverse direction. Because frame 689 is within frames 601-700 and thus within range 218C, search module 112 may find application launch content 216 and stop searching. As a result, search module 112 may not search the remaining ranges of index 214 (e.g., 218D-218N).

In some examples, it may be assumed (e.g., because of industry practice) that application launch content 216 is not within an initial range of index 214 (e.g., frames 1-100 of index 214). In these examples, search module 112 may not search the initial range of index 214 for application launch content 216. Search module 112 may define the sizes of ranges 218 by excluding the initial range from index 214 and then iteratively dividing the remaining frames in half. For example, if index 214 includes 800 frames and the initial range of index 214 includes frames 1-100 of index 214, range 218A may include 350 frames of index 214, range 218B may

include 175 frames of index 214, range 218C may include 88 frames of index 214, range 218D may include 44 frames of index 214, range 218E may include 22 frames of index 214, etc.

Experiments show that, in general, the Fibonacci-based algorithm outperforms the binary search algorithm. Experiments show that both the Fibonacci-based algorithm and the binary search algorithm can find launch application content in application content repository 110 faster than some existing solutions. Thus, usage of search module 112 in accordance with techniques of this disclosure may advantageously decrease TTID.

It is noted that the techniques discussed here may be combined with any other suitable technique or combination of techniques. As one example, the techniques discussed here may be combined with the techniques described in U.S. Patent Application Publication No. 2007/0071404A1. In another example, the techniques discussed here may be combined with the techniques described in WO2021141252A1. In yet another example, the techniques discussed here may be combined with the techniques described in U.S. Patent Application Publication No. 2009/0046776A1. In yet another example, the techniques discussed here may be combined with the techniques described in U.S. Patent Application Publication No. 2021/0012155A1. In yet another example, the techniques discussed here may be combined with the techniques described in Ben Pearson, “Measure Android App Startup Time,” Stack Overflow, October 28, 2015. In yet another example, the techniques discussed here may be combined with the techniques described in Android Developer, “How to improve application startup performance | Facebook application experience sharing,” Segment Fault, December 16, 2021.