Item 154
git_comments:

1. * * This class tests that methods in DatanodeDescriptor
2. * * Test that getInvalidateBlocks observes the maxlimit.
3. * * Licensed to the Apache Software Foundation (ASF) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. The ASF licenses this file * to you under the Apache License, Version 2.0 (the * "License"); you may not use this file except in compliance * with the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License.
4. remove from tree via iterator, if we are removing less than total available blocks
5. now if the number of blocks removed equals available blocks, them remove all blocks in one fell swoop via clear
6. insert into array ...
7. allocate the properly sized block array ...
8. iterate tree collecting n blocks...

git_commits:

1. **summary:** HADOOP-4483 Honor the max parameter in DatanodeDescriptor.getBlockArray(...). (Ahad Rana and Hairong Kuang via szetszwo)
   message: HADOOP-4483 Honor the max parameter in DatanodeDescriptor.getBlockArray(...). (Ahad Rana and Hairong Kuang via szetszwo) git-svn-id: https://svn.apache.org/repos/asf/hadoop/core/branches/branch-0.19@709024 13f79535-47bb-0310-9956-ffa450edef68

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
   description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method may discard the passed in array (and allocate a new array) if the number of elements returned by the iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.
2. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
   description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into

the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method may discard the passed in array (and allocate a new array) if the number of elements returned by the iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.
label: code-design

3. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method may discard the passed in array (and allocate a new array) if the number of elements returned by the iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.
4. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method may discard the passed in array (and allocate a new array) if the number of elements returned by the iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.
label: code-design
5. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method may discard the passed in array (and allocate a new array) if the number of elements returned by the iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.

6. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
   description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method may discard the passed in array (and allocate a new array) if the number of elements returned by the iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.
   label: code-design
7. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
   description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method may discard the passed in array (and allocate a new array) if the number of elements returned by the iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.
   label: code-design
8. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
   description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method may discard the passed in array (and allocate a new array) if the number of elements returned by the iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.
9. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
   description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method may discard the passed in array (and allocate a new array) if the number of elements returned by the iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-

reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.
label: code-design

10. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method may discard the passed in array (and allocate a new array) if the number of elements returned by the iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.
label: code-design
11. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method may discard the passed in array (and allocate a new array) if the number of elements returned by the iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.
label: code-design
12. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method may discard the passed in array (and allocate a new array) if the number of elements returned by the iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.
13. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method may discard the passed in array (and allocate a new array) if the number of elements returned by the iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This

INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.

14. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
    description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method may discard the passed in array (and allocate a new array) if the number of elements returned by the iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.
    label: code-design
15. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
    description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method may discard the passed in array (and allocate a new array) if the number of elements returned by the iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.
    label: test
16. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
    description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method may discard the passed in array (and allocate a new array) if the number of elements returned by the iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.
17. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
    description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method may discard the passed in array (and allocate a new array) if the number of elements returned by the

iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.

18. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method may discard the passed in array (and allocate a new array) if the number of elements returned by the iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.
19. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method may discard the passed in array (and allocate a new array) if the number of elements returned by the iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.
20. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method may discard the passed in array (and allocate a new array) if the number of elements returned by the iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.
21. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method

may discard the passed in array (and allocate a new array) if the number of elements returned by the iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.

22. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method may discard the passed in array (and allocate a new array) if the number of elements returned by the iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.
23. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method may discard the passed in array (and allocate a new array) if the number of elements returned by the iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.
label: code-design
24. **summary:** getBlockArray in DatanodeDescriptor does not honor passed in maxblocks value
description: The getBlockArray method in DatanodeDescriptor.java should honor the passed in maxblocks parameter. In its current form it passed in an array sized to min(maxblocks,blocks.size()) into the Collections.toArray method. As the javadoc for Collections.toArray indicates, the toArray method may discard the passed in array (and allocate a new array) if the number of elements returned by the iterator exceeds the size of the passed in array. As a result, the flawed implementation of this method would return all the invalid blocks for a data node in one go, and thus trigger the NameNode to send a DNA_INVALIDATE command to the DataNode with an excessively large number of blocks. This INVALIDATE command, in turn, could potentially take a very long time to process at the DataNode, and since DatanodeCommand(s) are processed in between heartbeats at the DataNode, this would trigger the NameNode to consider the DataNode to be offline / unresponsive (due to a lack of heartbeats). In our use-case at CommonCrawl.org, we regularly do large scale hdfs file deletions after certain stages of our map-reduce pipeline. These deletes would make certain DataNode(s) unresponsive, and thus impact the cluster's capability to properly balance file-system reads / writes across the whole available cluster. This problem only surfaced once we migrated from our 16.2 deployment to the current 18.1 release.

jira_issues_comments:

1. This fixes the getBlockArray method in DatanodeDescriptor to constrained the returned Block array to the maxBlocks values passed in.
2. **body:** Good catch on the bug. - Removing the elements form a collection one by one could be expensive. Also, we have max >= n in most of the cases. How about using the existing codes (i.e. blocks.clear() instead of e.remove()) when max >= n? - Could you also remove the white space changes, tabs and the trailing spaces? This will keep the codes have the same style.
   label: code-design
3. Good catch. This is a candidate for 0.19.1
4. **body:** Re: Removing the elements form a collection one by one could be expensive. Also, we have max >= n in most of the cases. How about using the existing codes (i.e. blocks.clear() instead of e.remove()) when max >= n? I believe that since the underlying container is a tree, even Collection's internal code needs to use the iterator approach to remove items from the data structure. Plus, in the standard configuration, where the heartbeat interval is set to 3 seconds, I believe max blocks <= 100. Better to stick to one code path in this instance. Re: Could you also remove the white space changes, tabs and the trailing spaces? This will keep the codes have the same style. Sorry, one my editors must to be set to tabs vs. spaces or vice versa. Am I correct in assuming that the convention for the hadoop codebase is spaces (instead of tabs) ?
   label: code-design
5. bq. I believe that since the underlying container is a tree, even Collection's internal code needs to use the iterator approach to remove items from the data structure. That is wrong. clear() removes all elements. You only have to set root = null, bq. Plus, in the standard configuration, where the heartbeat interval is set to 3 seconds, I believe max blocks <= 100. That's exactly we are removing the whole tree most of the times. > Am I correct in assuming that the convention for the hadoop codebase is spaces (instead of tabs) ? Yes, we are not using tabs in the source codes.
6. **body:** If we want to make the implementation to be more efficient, I have the following suggestions: 1. Store invalidateBlocks as an array or arrayList instead of a treeSet; 2. ReplicationMonitor makes sure that the size of invalidateBlocks does not go beyond blockInvalidateLimit; 3. getBlockArray does not need to worry about the number of invalidate blocks. It needs only to do an array copy and reset.
   label: code-design
7. **body:** Re: That is wrong. clear() removes all elements. You only have to set root = null, You are right. I see that TreeSet uses TreeMap as the underlying container, and, each iterator.remove causes a re-balance of the red-black tree. And, yes, looks like TreeSet.clear() set root = null, which is obviously speedier. I would still argue that under normal load scenarios (from my initial observations), the number of blocks in the Collection are <= 10, and since delete on the CLRS implementation of the Red-Black Tree takes O(log n), there might not be that much gained from the .clear() optimization. In the edge case where the system is under heavy load, I observed block count in excess of 1000. In this case, a partial removal of the blocks (based on the max blocks limitation) would still require the iterator.remove pattern. So perhaps in the long term, it might be better to replace the underlying data structure, as Hairong suggests. I guess it would be interesting to find out if the code (post patch) is a performance bottle neck or not before undertaking the more aggressive modifications.
   label: code-design
8. Revised version with .clear() implementation in cases where n == Blocks.size()
9. **body:** @Nicholas/Hairong: Are you folks saying that the approach adopted by this patch is not sufficient and it needs more changes to make it efficient?
   label: code-design
10. **body:** When I looked at the code related to block invalidation, I had a question whey invalidateBlocks were implemented as TreeSet, which require log(n) for both Inserting & removing. If we limit the size of invalidateBlocks to be no greater than blockInvalidateLimit, an array or arrayList would be more efficient. Otherwise, a LinkedList would be still be better than a TreeSet.
    label: code-design
11. **body:** +1 HADOOP-4483-v2.patch looks good to me. >Are you folks saying that the approach adopted by this patch is not sufficient and it needs more changes to make it efficient? Current fix is good enough for this issue. If there is anything we could do for better performance, we could do it in a separated issue.
    label: code-design
12. >If there is anything we could do for better performance, we could do it in a separated issue. I agree. Even Nicholas's suggestion is not necessary for this issue. Because this makes the good case even better and worse case even worse. Besides, the new patch covers the cases (n<available) & (n==availiable); If thee is a programmatic error that causes (n>available), invalidateBlocks may not get cleared.

13. Thanks Ahad for the patch. Changed patch to delete empty lines and to generate it using "svn diff" from the top level of the source tree.
14. **body:** Removed more empty lines from earlier patch.
    label: code-design
15. **body:** Please include a unit test.
    label: test
16. Upload a patch with a junit test.
17. +1 patch looks good.
18. Thanks Hairong. Sorry, I couldn't get to this in time.
19. Junit tests passed on my local machine: BUILD SUCCESSFUL Total time: 113 minutes 11 seconds Ant test-patch result: [exec] +1 overall. [exec] +1 @author. The patch does not contain any @author tags. [exec] +1 tests included. The patch appears to include 3 new or modified tests. [exec] +1 javadoc. The javadoc tool did not generate any warning messages. [exec] +1 javac. The applied patch does not increase the total number of javac compiler warnings. [exec] +1 findbugs. The patch does not introduce any new Findbugs warnings. [exec] +1 Eclipse classpath. The patch retains Eclipse classpath integrity.
20. Hairong, we also need a 0.18 patch.
21. Here is the 18 patch.
22. I just committed this. Thanks, Ahad Rana and Hairong Kuang!
23. Integrated in Hadoop-trunk #647 (See [http://hudson.zones.apache.org/hudson/job/Hadoop-trunk/647/]) . Honor the max parameter in DatanodeDescriptor.getBlockArray(...). (Ahad Rana and Hairong Kuang via szetszwo)