International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249 – 8958 (Online), Volume-9 Issue-5, June 2020

Automated Feature Performance Modelling of Inode Cache

Eshwari A Madappa, Swathi S, Mayank Agrawal, Suresh Anjaneyalu

Abstract*: Inode is one of the subsystems of WAFL(Write Anywhere File Layout) file system. Inode cache is a dynamic subsystem that is percentage factor of available memory. Based on different workflows and the datasets inode cache grows and shrinks. Based on the study of customer related issues it is found that deploying such a workload and datasets at the scale, that customers typically deploy and exercise inode cache for the whole duration of test is very challenging, considering quality assurance test typically focuses on multiple subsystems. Inode cache behavior differs with steady state versus performance disruptive workflows such as volume offline, volume online, volume migration and backup/vault use cases. Based on the behavior observed on the internal test systems it is found inode cache disruptive workflows are exercised only during certain stages but not repeatedly for the duration of test and also it is hard to find out which volume is experiencing performance issues due to inode cache invalidation/shrink/rewarning. In this paper, trying to exercise the performance behavior of inode subsystem like the way customer does and try to monitor and model the subsystem using automation. Here considering the different key attributes and typical operations that effect the inode cache behavior and some of the interested counters statistics that need to be monitored for analyzing the performance behavior of inode cache. Exercising inode cache operations requires constant focus on how the inode cache is performing. Repeat and Rerun some of the targeted workflows for inode cache population invalidation/ shrink operations at constant intervals to model the behavior of the inode subsytem.

Keywords: Inode Cache Grow, Inode Cache Invalidation, Inode Cache Shrink, Snap Mirror, Volume Migration (Move).

I. INTRODUCTION

WAFL is a block level file system, where files can be stored anywhere in the system. In WAFL meta data is stored inside the files [1]. An inode is one of the subsystems under WAFL. An inode is a data structure that stores the attributes of a file including its type, size and even where its data reside etc. Inode number is unique integer number assigned to a file when it is created[2][3]. The general structure of the WAFL inode subsystem is shown in Fig. 1, inode stores the information about other inodes, it is called Root Inode, where the location is fixed. Inode is smaller in size, it either stores the content of the file or it contains the pointer to the data file block or to the indirect blocks and forms the tree structure [4][5].

Inode has 2 different forms, in-core and on-disk forms.

1. **In-core WIP(Wafl_Inode):** An in-core inode is called WIP. It resides in a 4K page called inode page. Each inode page can hold 5-7 inodes depending on releases and platforms.
2. **On-disk Inode:** The on-disk inode holds the permanent state of an inode. Its size is currently 192 bytes.

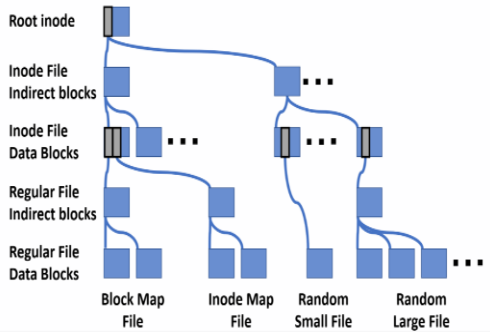

Fig. 1. Structure of WAFL Inode

Inode pages are linked together and maintained in the inode cache. For exercising the inode cache identified some of the key attributes, operations and counters to model and monitor the inode subsystem. Inode needs large volumes and multiple of such volumes and those volumes should have millions of inode created, it depends on the work specific file sizes some files are very small, some are very large and some files are all over the place. Inode cache is typically large for some volumes like NFS(Network File System) systems and volumes can be small for SAN(Storage Area Network) kind of systems, here focusing on the NFS systems. Inode cache behavior should change based on the operations that are performed on inode. Inode cache population and invalidation should happen over the time based on the volumes used and how it is impacted per volume, inode cache varies with different platforms as it is a matter of memory size.

The typical operations that impacted the inode subsystem are volume migration, backup, vault, and snapmirror. Typical key counters which are

Revised Manuscript Received on June 15, 2020.
* Correspondence Author
Eshwari A Madappa*, Assistant Professor, Electronics and Communication Engineering, JSS Science and Technological University, Mysuru, Karnataka, India. E-mail: eshwarinaveen@sjce.ac.in
Swathi S, M.Tech, Networking and Internet Engineering, JSS Science and Technological University, Mysuru, Karnataka, India. E-mail: swathisrinivas001@gmail.com
Mayank Agrawal, Member of Technical Staff at NetApp, Bangalore, Karnataka, India. E-mail: Mayank.Agrawal@netapp.com
Suresh Anjaneyalu, Member of Technical Staff at NetApp, Sunnyvale, US. E-mail: sureshkab@gmail.com

© The Authors. Published by Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP). This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)

interested in monitoring the behavior of the inodes are number of inodes on disk(inode_cnt), number of inodes to be recycled (inode_cnt. RECYCLE), inodes in memory (inode cache) and number of inodes that are pinned(pinned) vs unpinned(unpinned).
These are some of the important key factors for inodes based on which inodes behavior differs with the different operations performed.
The different key attributes, operations and counters used for the monitoring and modelling the inode cache is shown in Fig. 2.

Some definitions related to inode cache are:
Unpinned (inode) page: An inode page where all the wipes are in one of the following three states: UNUSED/ RECYCLE/NOSTATE.
Pinned (inode) page: An inode page where at least one wip is NOT in one of the following three states: UNUSED /RECYCLE/NOSTATE.
Inode Cache Grow: The process of creating inode pages as per the inode cache's sizing policy.
Inode Cache Scavenging/Shrinking: The process of freeing inode pages as per the inode cache's sizing policy.
Lazy Inode Invalidation : In a steady state system i.e. without any disruption to inode cache (i.e. no takeover/giveback, volume move, volume snapshot restore, file deletes etc) previously used volumes should invalidate inodes by switching the inode churning workload to different set of volumes in a lazy invalidation fashion assuming there is no memory pressure.

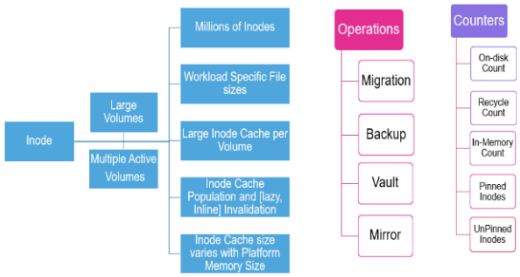

Fig. 2. Key Attributes, Operations and Counters for Monitoring the Inodes

II. PROBLEM STATEMENT

Considering one-year worth of customer related issues, data collected can be categorized into different ways based on the issue found. The 3 dominating categories are Performance, Feature Interaction and Race Condition. Here focusing on one category of issues that is performance.

For the customer data at high level, customer deploys larger datasets, workloads and the scale are at much higher rate with larger volumes and millions of data. They do both disruptive and non-disruptive workloads and operations based on the intent and requirement some of those are migration, tech refreshers, rebalancing workloads based on number of active and inactive volumes based on application demand. The scale of the data sets is in terms of 100s of TB. These are some of the key factors differentiated from testing done in internal systems and data deployed at customers system. As the customers are working at higher scale and datasets, could not able to find such issues in the internal systems.

Some customers related issues found in inode subsystems are, lazy invalidation failed to remove the inodes from the cache, front end changing workloads stresses the backend infrastructure based on the functionalities, they are indication of kernel panic as inodes are not keeping up with its operations, in case of v4 inodes system hangs during the upgradation and not able to service reads and writes operations.

III. PROPOSED SYSTEM

The general block diagram of how the inode subsystem is exercised as shown in Fig. 3. The system consists of the 2-node controller connected via HA(High Availability) pair, clients are connected to each node. Each node consists of different number of volumes, number of files created on each volume is different because of different volume size. When a new file is created in the volume, new inodes are created to store the information of a file, so inodes in the memory starts increasing. As the volume and the files in the system increases, traffic in the system also increases and reaches to a point where there will be no space available for new inodes to be cached, that's when lazy invalidation has to recycle the inodes which are not been used for longer time[6].

In some rare cases lazy invalidation could not kick out inodes and causes memory pressure in the system which effect the overall performance. In this paper, with different operations forcing the invalidation of inode cache is carried out. The first operation carried out is volume move operation[11], with this volume in the source node has been moved from source node to destination node. The second operation performed is snap mirror operation[12], which creates the snap mirror relationship between the volumes in the source node and destination node by replicating the volumes. The python and perl languages are used for automation for performing different operations[7][8][9].

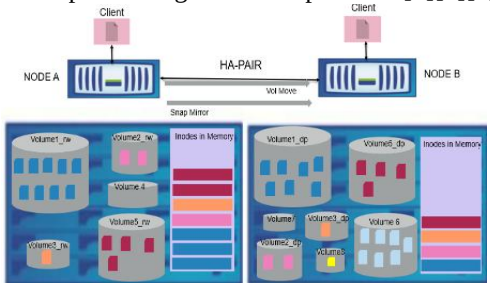

Fig. 3. General Block Diagram of How the Inode Subsystem is Exercised

IV. OBSERVATION FROM INTERNAL TESTING SYSTEM

Here are some of the observation seen from the internal test systems. The inode cache behavior in normal test systems is checked, collected some data of the inode cache behavior for the duration of 10 hours.

Observation from Internal System:

The number of inode and the inode cache count remains to be stagnant for the duration of test. Did not observe any inode cache

© Copyright: All rights reserved.

grow/population, shrink, invalidation and deletion happening in the system. In the same way the counter values like number of inodes that are pinned, unpinned, and recycled did not change in this duration of test.

The sample data collected from the internal systems is shown in Fig. 4.

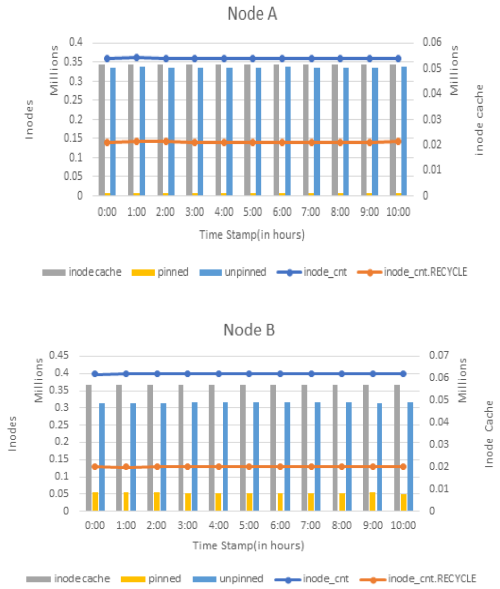

Fig. 4. Observation from Internal System

V. IMPLEMENTATION AND RESULTS

The inode cache in the system can be exercised by performing the focused operations. After the different operations have been carried out, now counter values are collected from perfstat tool [10], the results are used to plot the graph based on number of iterations carried for analyzing the inode subsystem. The graph consists of time stamp (in hours) in x-axis, number of inodes (in millions) in the y-axis, and inode cache (in millions) in z-axis.

1. The behavior and observation of On-Disk Inode Population on Inode cache:

Based on modelling work for exercising the inode subsystem, the inode cache grow/population is analyzed. For inode cache grow, the inode populator workload is used which will untar the files into the volume. The inode cache exercise more and more reads of on-disk inodes with files in the volume, with this inode cache starts growing/populating and it reaches maximum value and after that inode cache will be stagnant based on the amount of memory available in the system as shown in Fig. 5.

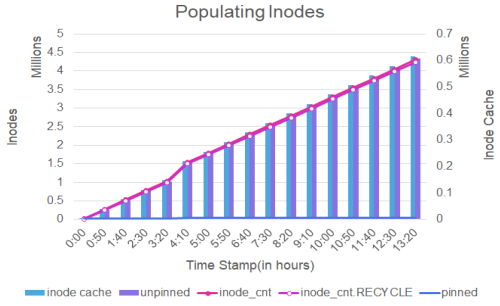

Fig. 5. Behavior of Inode Cache with Population

Observation from the Fig. 5 are:

- Number of Inodes (inode_cnt) and Inode cache increases with each iteration.
- The counters like inode_cnt.Recycle, pinned, unpinned also increases with the inode cache grow/population.
- The inodes in the system reaches to 4 Million.

2. The behavior and observation of Volume Migration on Inode Cache:

Once inode cache is populated, then considering those volumes with data populated and then perform volume move operation. Here the volumes have been moved from source node to the destination node. From the Fig. 6 and Fig. 7 after performing the volume move operation source node behavior is different from what is seen on the destination node. The cycle of iterations has been repeated for 6 times back to back so that consistency of how the inode cache impacted with the operation performed can be analyzed.

Observation of inode system with volume move operation:

Source Node:

The behavior of the source node with volume move operation is shown in the Fig. 6. Some of the observations are,

- The moment volume move operation is started, the inode cache for that volume tends to decrease, the number of inode cache goes down as volume move progresses and inode cache reaches the bottom, as when volume move operation completes volume decrease inode cache footprint.
- In parallel recycle of inodes (inode_cnt. RECYCLE) also reduces in memory.
- Number of pinned and unpinned inodes also sort of reduces as volume move will invalidate the inodes in the memory.

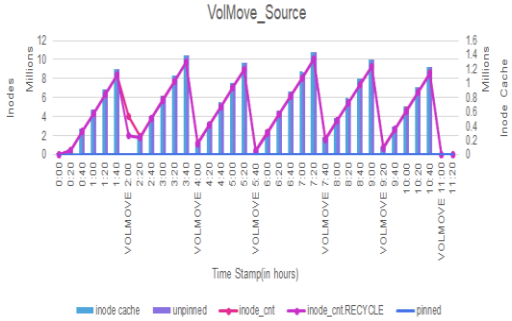

Fig. 6. Behavior of Source Node with Volume Move Operation

Destination Node:

The behavior of the destination node with volume move operation is slightly different as shown in the Fig. 7. Some of the observations are,

- On destination the behavior of the system is slightly different from what is observed in the source node, this is because it is just a volume move operation going on in the destination node and there is no front end reads and writes operation happening so no need to keep the inodes in the memory.
- The inodes in memory are less and it keeps populating as volume move operation progresses. As inodes reads happen on the destination inodes in memory

© Copyright: All rights reserved.

(inode_cnt. RECYCLE) and inode cache in the system increases.

- With the inode cache, pinned and unpinned counters also increases slowly with the volume move operation.

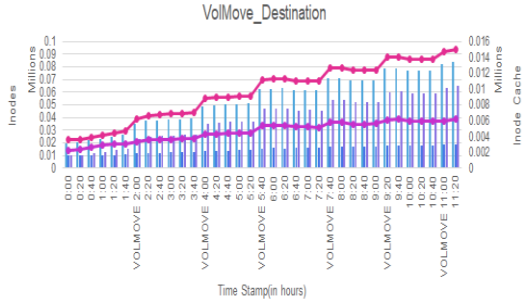

Fig. 7. Behavior of Destination Node with Volume Move Operation

3. The behavior of Snap Mirror Operation on Inode Cache:

Here by performing the snapmirror operation trying to exercise the inodes in the system. By creating the snapmirror relationship between the volumes in the source node and the destination node the replication of the volumes has been created.

Observation of inode system with snapmirror operation:

The different behaviour is seen on both the source and the destination node with the snapmirror operation. Some of the observations are,

Source Node:

- With the population the number of inodes (inode_cnt) and inode cache increases
- After reaching maximum value, the number of inodes in cache remains to be stagnant at the source.
- Similarly, the inode cache pinned and unpinned counters also increases as shown in the Fig. 8.

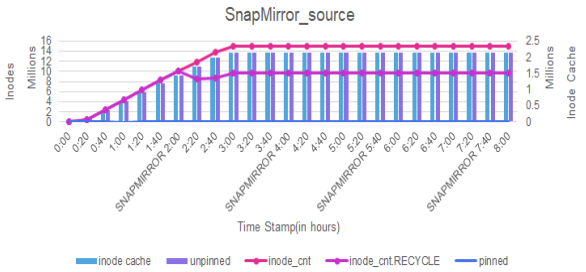

Fig. 8. Behavior of Source Node with Snap Mirror Operation

Destination Node:

- As the new inodes are coming from the snapmirror_source the inodes populates slowly over the period.
- With increases in the inode cache, pinned and unpinned counters also increases as shown in the Fig. 9.

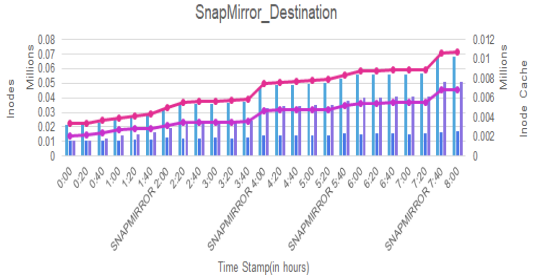

Fig. 9. Behavior of Destination Node with Snap Mirror Operation

4. The behavior of forced Inode Cache Shrink:

For the internal test system with smaller datasets, also wanted to check how we can force the inode cache shrink to happen without performing any operation. To depict that used on-box flag to force the shrink to happen over the period. The cycle of iterations has been repeated for 3 times back to back so that consistency of how the inode cache is invalidated with the flag can be analyzed.

Observations from forced inode cache shrink:

Some of the observations are,

- Inode cache increases with population with lesser scale of 3 million inodes in the system.
- By using the on-box flag (icache_ inode_reclaims) which will reclaim all the inodes populated till the time

- Inode cache starts invalidating the inodes and it decreases its value with the flag
- Overall, the observation is that we can force the shrink to happen by having that flag as shown in the Fig. 10.

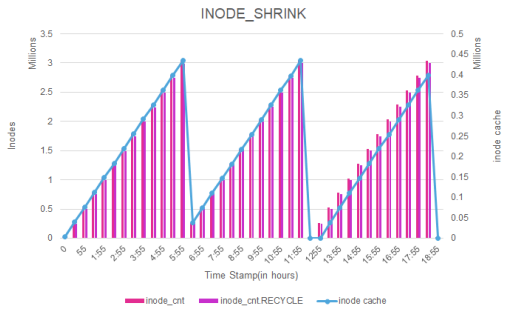

Fig. 10. Behavior of Inode Cache with Shrink

Overall behavior observed by exercising the inode subsystem is that, doing a lot of inode population as well as volume move and snapmirror operation, which sort of building a forced feature interaction in the system. Which means inode cache shrink is working when volume move operation is carried and when snapmirror operation is carried. By performing different operations, system behavior and performance related issues within the system can be analyzed.

VI. CONCLUSION

With the modelling work carried for analyzing the behavior of the inode cache, forced feature interaction happening in the system that is inode cache growing and shrinking with the operations
like volume

© Copyright: All rights reserved.

move and snapmirror. With the focused performance centric automation can induce inode cache behavior dynamically by constantly polling the key performance attributes/counters along with targeted operations that affect inode cache.

FUTURE SCOPE

Open source tools like Grafana, Kibana can be used for the charts and log analysis. The idea of testing can be used to expand to other subsystems of WAFL such as snapshots, front-end reads/writes and various other back-end operations.

ACKNOWLEDGEMENT

My sincere thanks to our Principal, HOD (Department of ECE) of JSS Science and Technological University for the support given to do the project in the company. I would like to thank my guide Mrs. Eshwari A Madappa for guidance and support provided. I would like to thank my manager of NetApp Private Limited, Mr. Renuka Puttappa, Mr. Suresh Anjaneyalu and Mr. Mayank Agrawal for the encouragement and support provided. I thank the whole team for the support of technical guidance.

REFERENCES

1. https://www.usenix.org/legacy/publications/library/proceedings/osdi99/full_papers/hutchinson/hutchinson_html/node3.html
2. https://netapp-world.blogspot.com/2014/01/inodes-in-netapp.html
3. http://www.linfo.org/inode.html
4. https://www.wikiwand.com/en/Write_Anywhere_File_Layout
5. Dave Hitz, James Lau, and Michael Malcolm. “File system design for an NFS file server appliance”. In Proceedings of USENIX Winter 1994 Technical Conference, pages 235–246, Jan 1994.
6. Ram Kesavan, Rohit Singh, Travis Grusecki, Yuvraj Patel,” Algorithms and Data Structures for Efficient Free Space Reclamation in WAFL”, File and Storage Technologies (FAST’17) February 27–March 2, 2017
7. Chan Bernard Ki Hong, Perl 5 Tutorial, First Edition,2003
8. Ceder, V.L., McDonald, K., and Harms, D.D, The quick Python book (p.335), Manning, 2010.
9. Sanner, M.F, Python: a programming language for software integration and development, J Mol Graph Model, 17(1), pp.57-61, 1999.
10. https://www.sysadmintutorials.com/tutorials/netapp/netapp-perfstat-collecting-performance-and-statistics/ https://library.netapp.com/ecmdocs/ECMP1368017/html/GUID-C5685C7A-063C-43EC-B17E-FB7B352D7023.html
11. https://library.netapp.com/ecmdocs/ECMP1635994/html/GUID-98E2BBA2-2A4F-4261-A390-9A712AB78761.html

AUTHORS PROFILE

Eshwari A Madappa Assistant Professor at Sri Jaya Chamarajendra College of Engineering Mysuru, Dept of Electronics and Communication. JSS Science and Technology University. She has completed her Bachelor of Engineering in JSSATE Bangalore and Master of Technology in Power Electronics stream in RVCE Bangalore.

Swathi S MTech student at Sri Jaya Chamarajendra College of Engineering Mysuru in Networking and Internet Engineering, Dept of Electronics and Communication. JSS Science and Technology University. She has completed her Bachelor of Engineering in GSSS Institute of Engineering and Technology, Mysuru

Mayank Agrawal Member of Technical Staff at NetApp India Ltd. He is having 10 years of experience in various storage and cloud technologies with QA automation. He has completed his Bachelor of Engineering from RGPV Bhopal and Master of Technology from BITS Pilani in Software Systems.

Suresh Anjaneyalu Member of Technical Staff at NetApp Sunnyvale, US. He is having 20 years of industry experience. He has completed his Bachelor of Science and Engineering from MSRIT Bangalore.

© Copyright: All rights reserved.