# A Prolegomenon for Improving Observability in iRODS

**Arcot Rajasekar**
School of Information and Library Science
University of North Carolina at Chapel Hill
rajasekar@unc.edu

## ABSTRACT

Observability is an emerging practice for measuring and interpreting the pulse of complex and distributed software systems. Software developers and IT teams need to understand when and why there is an abnormal behavior, how to mitigate that within a short time. With the acceleration of complexity, scale, and dynamic architectures coupled with automatic software patching, malicious intrusions and system breakdowns, high throughput system need more than to react to events but proactively predict and mitigate anomalous behavior before it happened. Challenges due to multiple combinations of things going wrong, and sympathetic reinforcements of faults can make it hard to track how errors are manifesting and how a system is behaving. Observability couples the ability to capture runtime telemetry with a visual and reasoning system that can detect and pinpoint abnormal behavior deep in the system before it can affect the performance of the whole system. Introduced first in control theory, observability is a measure of how well internal states of a system can be inferred by knowledge of its external outputs. The iRODS software system is not only a very complex and dynamic system it is also being increasingly deployed with other complex software on distributed infrastructures that rely on its high throughput and reliability. An introduction to the concept of observability in iRODS would be very helpful for making sure that a deployed iRODS installation is operating in an optimal manner. Moreover, with observability built into iRODS, one can find operational anomalies in systems that are relying on iRODS and help mitigate them. The iRODS system already has a very strong measurement capability through its logging. Enhancing its capability with tracing, session replay, learning and analysis systems that can provide actionable insight would take iRODS to the next level of performance and resilience. In this paper, we look at various ways one can enhance iRODS to become a highly 'observable' system.

### Keywords

iRODS, observability, metrics, logs, traces

## INTRODUCTION

Observability[1, 2, 3] is the ability to understand the inner workings of a system by observing the external behavior. Originally described in control theory[4, 5], it is being increasingly being adopted by software engineers and system administrators to monitor system malfunctions and if possibly proactively preempt any such malfunctions. As the complexity of systems increase due to using multiple chained services, parallel and distributed processing, and multi-cloud environments there is a concomitant increase in pressure to maintain operational viability and graceful degradation of services. Hence reliability engineering and DevOps[6] are increasingly looking at observability as a means to provide better and more reliable and resilient services. With distributed data, computing and higher reliance on networking problems can crop up at multiple levels and become apparent only at a later stage as critical systems start to fail and there is a chain reaction effect leading to eventual shut down of the system. Even small failures can be a problem which can crop up because of a sequence of events in a distributed system and hence need to be captured and solved and made resilient in the future.

*iRODS UGM 2021* June 8-11, 2021, Virtual

In computing systems, reliant on distributed hardware, multiple networking interconnects (from LANs, WANs and clusters) and distributed software services, developed by different vendors, reliability of the whole system to operate at an optimal level is of high concern. Observability provides a proactive way to monitor a large system and develop strategies in case of failures. In software and hardware systems, observability is achieved by measuring system critical components through various sensors and telemetry systems for hardware and logs and metrics and traces for software systems. Observability is implemented using a combination of instrumentation methods including tools, such as Grafana[7], OpenTelemetry[8], Splunk[9], SigNoz[10], etc. Observability empowers cross-functional teams (IT Admins, system developers, application engineers, managers) to identify problems before they even manifest or become unmanageable. Observability helps increase performance, availability, resiliency and user satisfaction by enabling one to realize what is slow, what is broken, and to quickly figure out what needs to be done to improve performance.

With the increased complexity of software system, it is highly becoming crucial to apply observability principles in order to run a viable enterprise. An innocuous update in an obscure software package can have a cascading effect. Finding problems and correcting them can be a nightmare and going beyond that, predicting failure and degradation of services can be rewarding but highly challenging. Moreover, with privacy and security considerations becoming increasingly important, observability can be key factor in finding anomalies and intrusions in a highly distributed networked system.

The integrated Rule Oriented System (iRODS)[11, 12, 13, 14, 15] is an open source software used for large-scale data management software used by research, commercial, and governmental organizations worldwide. It virtualizes storage system and presents a unified view of all digital artifacts stored under its management. The fundamental difference that iRODS brings to distributed data management is its provision of a rule-based operational characteristic Users, administrators and system developers can write complex rules to govern the management of their data throughout its data life cycle. These rules can apply from a single file-level to the whole system level. The application of the rule is governed by conditions and can be applied on an event, periodically or at any arbitrary time by the user or administrator. These rules govern all aspects of the data management from file ingestion to storage, from access to deletion from the system. Indeed, one can override all default and built-in operations implemented in the iRODS system to completely perform an entirely different set of operations that is suitable for the enterprise. Hence the iRODS system provides a far more challenging environment for performing observability compared to a static system whose actions are mostly defined as per the original developers' intent.

iRODS implements a complex and dynamic system that spans multiple services that run on distributed hardware across the wide area network. It is also being increasingly deployed with other complex software on distributed infrastructures that rely on its high throughput and reliability. An introduction to the concept of observability in iRODS would be very helpful for making sure that a deployed iRODS installation is operating in an optimal manner. Moreover, with observability built into iRODS, one can find operational anomalies in systems that are relying on iRODS and help mitigate them. The iRODS system already has a very strong measurement capability through its logging. Enhancing its capability with metrics, tracing, session replay, learning and analysis systems that can provide actionable insight would take iRODS to the next level of performance and resilience. In this paper, we look at various ways one can enhance iRODS to become a highly 'observable' system.

## MONITORING FOR OBSERVABILITY

Monitoring observability relies on three main concepts: metrics, logs, and traces. These three are considered the pillars of observability. One needs to add monitoring and telemetry to instrument both the software and the hardware to gather useful information that can be used to find if the system is running in a sub-optimal manner or to predict potential problems down the road. When a fault is detected, analyzing these three measures will be helpful in pinpointing areas of concern.

**Metrics** are the operational characteristics that are collected to define what is the optimal performance for a system. These are derived over time based on system performance. Types of metrics include response time for a service (micro-

service), uptime and downtime of systems, number of requests serviced during a period of time (e.g. concurrency), load of the system including cpu, memory and network utilization, etc. In many cases, service level agreements (SLA) or service level objectives (SLO) provide the metrics that need to be met for providing required performance. Metrics provide the baseline to which one can compare operations at the current level to find any anomalies. Moreover, metrics provide a means to detect degradation of service. More metrics is defined and continuously monitored, the better will be the reaction time to solve problems or preempt them before they become critical. Service level indicators (SLI) provide the measurement that detect the service behavior at the current time. Comparing SLI against SLO or SLA provide a measure of the performance. SLI can be used not just to measure if the system is meeting current requirements but it can also be used to find slack in the system that can be utilized to provide more service without upgrading the system. As the SLI shows a degradation against an SLO or SLA, then solutions to improve service – such as bringing an additional server online or performing better load balancing – would be needed. Similarly, Key Performance Indicators (KPI) are also necessary metrics that capture data about crucial performance measurements. KPIs provide the necessary baseline data that can be used for strategic and operational improvement and to provide an analytical basis for decision making. KPIs helps by setting targets that capture the most important critical metrics for the success of a system and help focus attention on what matters most. KPIs and SLIs together help to assess progress towards optimal performance results. They are critical for tracking efficiency, effectiveness, quality, timeliness, governance, compliance, system performance and resource utilization.

**Logs** are basically outputs written by software procedures to record their operations. Hence in a well-written procedure or micro-service, print statements to a log device (either a log file, or a message to a queuing system or to a remote recorder) capture the current status and actions of the procedure. Since they are written by the system developer, these can be highly structured and provide enormous details about the status of the system as well as about the operation being performed. For example, key log outputs can be written when a process enters a micro-service and when it exits a micro-service. In such a case, some entry data can be part of the log, and because there can be more than one exit point for the micro-service, details of such exit, normal or abnormal, can be captured. In particular, these two entry-exit logs will have fine-grain timestamps and provide a measure of how long the process took to perform this micro-service. Log analysis can help to uncover abnormal, unpredictable and emergent behaviors by procedures. Logs are easy to generate provided the system developer has instrumented the code appropriately. In many systems, there may be multiple levels of logging and the administrator can choose the level that would be comfortable for the system. Since a logging operation take a finite amount of time, intrusive logging by themselves can slow down the function of a procedure. Hence one needs to be aware of how much of logging is desired to monitor the system against the degradation of service that is introduced by the logging process. Distributed logs (logs gathered from distributed systems) are normally gathered at a central location and analyzed. Such analysis provides a means of finding the reliance of one system over other and can be used to drill down when an operation takes more time than normal.

**Tracing** is the concept of capturing the journey of a user request from start to finish and to find what are all the services and components that are involved in answering that request. Particularly in a distributed system, tracing provides a means to find all the nodes of the system through which the service request 'touches' on its way to completion. Hence trace data is more complicated than logging but rely on logging to find traces of all actions being performed by the system. Traces provides a snapshot into system behavior such as which component took the longest or shortest period of time, or whether specific underlying functions resulted in errors. Traces provide context to a user request and can be used to find what is really happening when a user request is entered into the system (by user request, we mean the high level request that a user makes – e.g. for Google it can be the search for a keyword, or a request for a routing path to the Map application, or in the case of a bank, it can be a request to find the account balance or perform a money transfer or pay a bill). In particular, tracing helps to analyze individual user calls and understand the sequence of (often distributed) steps that are taken to return a result. Tracing also helps to analyze how the calls behave for different users, and also when the results are successful or failure.

Journeys provide the sum-total of all activities that is performed during a user session. A journey can have multiple sub-journeys. Each journey can be made of several paths which can be parallel in a distributed system. A journey

captures timing, possibly call and return expressions, status code and anything else that an Observer deems to be necessary. Journey can be abstracted into templates and help find bottlenecks and errors so they can be fixed and optimized. By analyzing trace data and journeys, one measures not only the overall system health, but also by looking at similar traces can pinpoint breakdowns or slowdowns, find bottlenecks, identify issues that can lead to performance degradation and also provide a means to prioritize important code snippets for optimization and improvements. For example, tracing can help find common bottlenecks (e.g. a database call) that is used by different traces and hitting similar slowdown.

To perform better observability all three of these measures – metrics, logs, and traces – are needed. In many cases data from multiple of these measures will be used in a dashboard to provide a clear picture of what is happening in a system. The power of observability comes from a unified approach – individually these measures give only a partial picture – in taking advantage of these three types of measurements to find the health of a given software system. Observability dashboards gain insight by coalescing data from all three measures and using them to find anomalies and problems that need to be solved.

Visualization and analytics play a key role in observability. Visual analytics, with human-oriented graphs, are used by observability dashboards to provide the status of a system in real-time. Visual cues, color coding and other tricks are normally used to pinpoint abnormalities. Artificial intelligence, machine learning and predictive and post-mortem analyses are key tools in making observability a success. Continuous analytics, adaptive learning, deep analytics are increasing being used to find anomalies before they become life-threatening problems to the systems health.

## CURRENT STATUS OF OBSERVABILITY IN iRODS

The iRODS system is a complex software that spans multiple hardware as well as software services. The rule-based system – using micro-services – makes it harder to define common patterns of actions as each installation can have its own idiosyncratic rule applications.

Even though observability is not currently being performed in any iRODS system, there is a rich support for eventually providing a good measure of observability capabilities. The iRODS system is a mixed bag when considered from observability point of view, it has some aspects that are vital to perform observability, but also lacks other crucial measures and tools to become successful in observability. There are three types of measurements one can identify in iRODS that can be helpful for observability.

### Server Logs

Server logs are collected in each distributed storage server and the iCAT-enabled server in the form of a server log file. These contain mainly print statements from various functions and procedures in the iRODS system, crucially from the micro-services. In iRODS there are levels of server log monitoring and the site administrators can throttle the levels as needed to capture coarse or fine-grained information.

### Audit Trails

The audit trails provide a different type of logging, but mainly tied to the action performed on the data objects stored in the iRODS system. For example, audit trails captured the information about when someone created a file, who wrote or modified the file and who accessed the file. Timestamps are also recorded. Audit trails collect user information on triggered actions. One can use them to recreate traces from a data object viewpoint about what happened to it during its lifetime.

### Status Metadata

This information comes from the iCAT metadata catalog. iCAT stores persistent information in its database which can be used to analyze different types of metrics. For example, the size of the file and the time it took to access it can be used together to find whether the system is operating in an optimal fashion.

At the current level, iRODS is not doing any observability. One can view the current support in iRODS is more towards system monitoring activities with no observability functions. Server Logs, Audit Trails and Status Metadata in iRODS provide a strong and stable foundation for implementing observability. Use of policies, rules and microservices provide one more level for gaining information to perform observability.

## IMPROVING OBSERVABILITY IN iRODS

A concerted effort is needed to upgrade iRODS functionality to perform true observability. At the current stage iRODS is lacking critical features and functionalities that hinder it from being a viable observability system that can optimize performance, predict problems, and isolate malforming processes. In its current state, iRODS provides limited log and trace measurements. Missing functionalities include metrics and journeys, apart from implementation additions for more useful logs and traces. Missing tools include visualizations, dashboards, and analytics. We believe that metrics and improved logs, traces and journeys can be implemented in iRODS without any change to its software stack, thanks to the rule-based micro-services paradigm of its underlying architecture. Additional modules for visualization and run-time analytics are needed to be developed (or adapted from open source implementations) to provide administrators insights into its day-to-day operational characteristics.

### Logs and Traces

One big advantage to implementing iRODS comes from its current provision for server logs, audit trails and status metadata. One can use these three measurements that are already implemented in iRODS to perform log-based operations and tracing operations. But there are some crucial issues that need to be solved before we can do that. An important bottleneck is the distributed nature of the server logs. Server logs are stored at individual storage servers and are not synchronized enough to be a useful form of log and trace measurements. Hence, centralizing these with time synchronizations would be the first step that one needs to do to take advantage of these outputs from iRODS procedures and function calls. As mentioned earlier, audit trails are data-centric rather than session centric. Hence their value is doubtful when using them to perform traces or to use log activities to find sub-optimal operations. They may provide a valuable 'journey-type' information about how multiple data objects are being used – but those journeys may not be key to observability. Analyzing them may still be useful and can provide insights into pre-staging, load balancing and replication.

Status metadata can be used as metrics information. But the type of status that is logged in the iCAT may prove to be useless from the observability point of view. Even though iCAT captures a rich amount of metrics in these metadata, most of these are data object related and/or user and resource related. They may not provide insights into operational data necessary to define metrics measurements. Some possibilities still exists: location of objects, length of objects, limits on parallel transfer operations and such can be useful in defining some lower and upper bounds of operational metrics.

### New Metrics Measures for iRODS

Operational system level metrics are necessary to be developed for iRODS in order to provide the baseline for application performance monitoring (APM). APMs provide critical checks about whether the system satisfies SLA contracts, meets SLO performance standards, identify bugs and potential issues, and provide flawless user experiences via close monitoring of IT resources and process timelines. APMs are also needed to increase mean time between problems and reduce mean time between resolution of such problems. Continuous monitoring and comparing against standard metrics is crucial towards proactive remediation. APM with metrics can be used to provide just in time alerts and perform simple analysis that can help to identify possible problems in the future.

There are several metrics measurements that can be implemented to help monitor the health of any iRODS implementation. Many of these metrics can be standardized across implementations, but others need to be defined per installation. Moreover, these measures themselves can change over time as system gets loaded and/or modified and upgraded. We provide one such set of metrics that we feel is important as a starting point for iRODS. This list is by no means an exhaustive list and more will be added as the observability capabilities are improved in the future.

Many of these can be considered as SLAs or SLOs that provide KPIs. Continuous monitoring will provide the needed SLIs to take advantage of these metrics.

- CPU/Memory Usage (max, min and average)
- Network Traffic (max, min, optimal connections and loads)
- Database Load (concurrent usage, speed for critical queries)
- Error Types/Rates (current errors that are not showstoppers)
- Request rates (connections per unit time, max, min, average)
- Response times (mean, max, min)
- Bandwidth/Throughput (volume and number of files)
- Concurrent Connections (external load on the system)
- Number of instances/threads (internal load on the system)
- Microservice/function usage/time (per micro-service metrics)
- Uptime, Restarts & Availability (system SLOs)
- User Experience KPIs: SLAs, SLOs for critical user requests
- Software Module KPIs: SLOs are each major and some minor software modules.

The above metrics are needed to measure against run-time performance of the iRODS system. Some of these measures are formulaic as they depend upon the data object size or system configuration size (e.g. how many cores). But it is a good idea to lay down some metrics that can help one to see where one is meeting important KPIs. As mentioned earlier, KPIs help in setting targets that capture the most important critical metrics for the success of a system and help focus attention on what matters most. KPIs and SLIs together help to assess progress towards optimal performance results and user experience. They are critical for tracking efficiency, effectiveness, timeliness, system performance and resource utilization.

#### Journeys for iRODS

Journeys are important to find critical modules, performance of modules as well as to provide insight into how a user request is addressed. Journeys use traces but build a profile of a session that is more informative than a trace. In the case of iRODS to define a journey one needs to first address the complexity of the distributed tracing (DT) that needs to be analyzed in order to define the actions performed by a user request. Chaining of services (including microservices) and peer-to-peer connections across distributed systems makes it hard to trace the activities of a session but is critical for performance monitoring. DT helps identify bottlenecks across dynamic and heterogeneous infrastructures on which an iRODS system is executed.

In order to define journeys, the current server logs need to be enhanced with more operational reporting. One may reduce the burden in developing this capability by only increasing the reporting of crucial modules with more useful data that can help when creating the journey profiles. As mentioned earlier, a critical problem is the distributed nature of the server logs in their current implementation. One may need to consider methods to centralize them in real-time so that one can take advantage of them to analyze real-time abnormalities but also to develop prototypes for journeys that can be compared for anomalies.

To take advantage of the concept of journeys in iRODS the following types of analysis need to be implemented:

- Distributed Transaction Monitoring and Analysis
- Create User or Application Profiles
- Define Patterns and Templates of Journeys and Sub-journeys
- Latency optimization
- Failure Models
- Alternate Pathways
- Service Dependency Analysis
- Critical Path Analysis
- Root Cause Analysis

Providing these analyses will help in building journeys and in timing their performance and finding deviance from optimal levels.

#### Observability Analytics for iRODS

At the high level, there are four types of analytics that are relevant for observability that can be utilized to gain insights from traces, logs, metrics and journeys.

- Predictive analytics can provide insights into what is likely to happen based on historical data. Using data mining, statistical modeling and machine learning predictive analytics help in defining probabilistic future outcomes. Analyzing past patterns and trends one can gain insights to induct what might happen going forward and, in doing so, inform about setting realistic goals, effective planning for failure as well as improvements, managing performance expectations and avoiding risks. Predictive analytics play a key role in observability as one can look at various patterns in journeys and traces to figure possible failures down the road. In iRODS such trends can be used to identify network degradation, resource over utilization, and take actions to overcome them.
- Diagnostic analytics (also called post-mortem analysis) provide a means to figure what adverse event happened and help to find out why it happened. Forensic analysis of traces, journeys and logs help in figuring out the root cause of a failure after it has happened. It can also provide insight into how one can mitigate such failures in future through corrective actions. Diagnostic analytics use data aggregation and data discovery methods to find historical patterns. Visual analytics also play a key role in these types of analytics. Summary statistics, clustering, pattern tracking, and regression analysis are some of the ways one can find patterns and failure mode backtracking in the data and measure performance.
- Descriptive analytics provides insights and reports on the inner working of the system. They can be used to aggregate performance metrics as well as report on events. They can be used to find the most used micro-service and such independent of traces and journeys. An important use of descriptive analytics is to check on the value of the metrics defined for the system and fine tuning some of them for optimal performance. User profiles, resource profiles, bandwidth utilizations are some of the key factors that can be aggregated and used towards improving performance.
- Prescriptive analytics helps you how to avoid certain adverse event from happening. It can tell what, when and why something can happen. It is in a sense like game playing. One can identify different scenarios and simulate actions and see how the system behaves under such circumstances. For example, one can use prescriptive analytics to find when a particular storage system can break and under what loads can we expect worse performance beyond what is promised under an SLA. Prescriptive analysis can be used in conjunction

with the other three types of analytics to provide valuable insights to make evidence-based reasoning about future events. For example, by tweaking a journey, one can see under what scenarios the journey will adversely affect the outcome of a request. Predicting fluctuations, perturbations, assessing upgrades are some of the uses in iRODS for such analyses.

Observability uses multiple types of algorithmic solutions to realize the above three types of analytics. Here are some examples of such types of analytics that will help perform observability in iRODS.

- Statistical analytics is used to analyze metrics data for informative nuggets. Max, min, median, mean, standard deviation, etc. can provide valuable insights. Statistics can be used to define norms, SLAs and expected outcomes and latencies. Hence is a way to define important KPIs.
- Graph Analytics use traces and journeys to find patterns. Pattern analysis. Critical nodes and Most used nodes. They can be used to perform predictive as well as prescriptive analysis. Graph analysis is also a very useful way of finding outliers and anomalies in data. In the iRODS context, couple of uses can be seen in pre-staging and delayed processing.
- Visual analytics is another form of graph analytics and is very useful when performing temporal and spatial data analysis. They are also used to fusing multiple layers of data together and provide insights that can stand out in a visual setting. One can find trends, collinearities, correspondences, correlations and clusters, and anomalies using visual analytics.
- Text Analytics: These can be used effectively when analyzing log files. They will also be effective in creating journeys and analyzing them. One can use contextual data of journey to define dynamic slicing and figure out repeatable experiences with possible minor deviations.
- Data Mining, Machine Learning and Deep Learning will help us to learn good and bad patterns. For example, one can teach a neural network about multiple journeys, both successful and failed journeys. Then when in real-time a journey is progressing, the neural network model can be used to predict success and failures based on the pre-path of the journey and take corrective actions to mitigate the risk of failure.

## CONCLUSION

This paper provided a glimpse of how one can perform observability in a complex system such as the iRODS. Observability is becoming increasingly important because of complexities of the applications as well as need for high availability and throughput by the user community. Observability can be used as a means to monitor the system continuously and correct them on the fly. Observability can also provide insight to developers and administrators on how system performance and user experience can be improved. Current status of iRODS provides some rich measures that can be effectively used to perform observability. Server logs, audit trails and iCAT metadata can be the basis for assembling an observable system metrics and traces. But there is clear need for improvements. Metrics can be improved, and server logs can be centralized and improved for utility as a measure for observability. The concept of journeys is new for iRODS as its current metadata caters mainly to data-centric life cycle. Plugging in concepts for analytics can make iRODS more effective for future users. iRODS is used as an enterprise system and hence it needs observability tools to make it a world-class enterprise system.

## REFERENCES


[1] Burns, J. The complete guide to observability. Lightstep Research White paper. `https://go.lightstep.com/rs/260-KGM-472/images/observability-guide.pdf`

[2] Trivedi, Tapan. (2018). Controllability and Observability. 10.13140/RG.2.2.33290.62407.

[3] Vidal, R., Chiuso, A. Soatto, S., & Sastry, S. (2003, April). Observability of linear hybrid systems. In International Workshop on Hybrid Systems: Computation and Control (pp. 526-539). Springer, Berlin, Heidelberg.

[4] Kalman R. E. On the General Theory of Control Systems, Proc. 1st Int. Cong. of IFAC, Moscow 1960 1481, Butterworth, London 1961.

[5] Kalman R. E. Mathematical Description of Linear Dynamical Systems. SIAM J. Contr. 1963 1 152.

[6] Jabbari, R., Ali, N., Petersen, K., Tanveer, B. (May 2016). What is DevOps?: A Systematic Mapping Study on Definitions and Practices. Proceedings of the 2016 Scientific Workshop.

[7] Grafana: The open Observability platform. `https://grafana.com`

[8] OpenTelemetry. `https://opentelemetry.io`

[9] Splunk: Observability for DevOps. `https://www.splunk.com/en_us/observability.html`

[10] SigNoz: Open Source APM. `https://signoz.io`

[11] iRODS: Open Source Data Management Software. `https://irods.org`

[12] Xu, H., R. Moore, A. Rajasekar. A Method for the Systematic Generation of Audit Logs in a Digital Preservation Environment and Its Experimental Implementation In a Production Ready System, iPRES conference, November 2015.

[13] Moore, R. W., A. Rajasekar, M. Wan. iRODS Policy Sets as Standards for Preservation, US-DPIF'10 Proceedings of the 2010 Roadmap for Digital preservation Interoperability Framework Workshop, 2010.

[14] Moore, R., A. Rajasekar, M. Wan. Policy-based Distributed Data Management Systems, OR'09, Atlanta, Georgia, May 2009.

[15] Rajasekar, A., M. Wan, R. Moore, W. Schroeder, Federation of Rule-Based Data Grids, Workshop on Digital Archive Preservation and Sustainability, September 2008, Baltimore, Maryland.