Cloud Native at the Edge: High-performance Edge Inference with RedisAI

arm

A Project Cassini Reference Implementation

White Paper

Index

Real-time analytics and data insights at Edge

Analysis of data collected from devices and resources such as sensors and edge devices is growing rapidly among industrial customers. Image processing, anomaly detection for predictive maintenance, and log analysis are some of the most popular use cases for edge analytics. Several reasons make running data analytics on edge devices attractive:

1. Analyzing data on edge devices can save money and resources by offloading the computations from cloud.
2. The latency of sending data to the cloud for analytics could be high, particularly when processing real-time events. Performing inference on real-time data may require the inference engine to be as close to the data source as possible.
3. Reliability of network connections can be an issue for IoT platforms, hence making on-device inference and decision making important in critical scenarios.

Challenges

Industrial IoT market is expected to experience the growth of 15.2% per year for the forecast period of 2020-2027 (Data Bridge Market Research, 2020), which is mostly driven by machine learning, AI, and data analytics. However, still most data in IIoT environments are not currently exploited for analytics, and huge number of devices are not even linked up to a sever in the cloud.

While many of applications such as assembly line redesign will take place in the cloud, a substantial number of tasks—predictive maintenance based on real time data, product consistency based on computer vision, or energy efficiency—will be performed locally at the edge. The money that industry can save by edge analytics such as predictive maintenance could be huge. Some mining companies have reported an estimated loss of $30k per hour in downtime. This is worse in automakers industry, which is $20k-$30k per minute (Altman, 2019). With more than 180 billion chips, Arm will play a big part in integration of the IoT functionality into devices for edge analytics.

Running analytics on edge cuts cost and latency while increasing reliability. Still, owners will have to balance their analytics needs with the core hardware and software capabilities. In general, IoT devices have limited computational power with lower resources, and many of the on-the-shelf analytic tools are not designed with an eye on running on small devices. Therefore, finding right tools and techniques to run applications on the edge could be challenging.

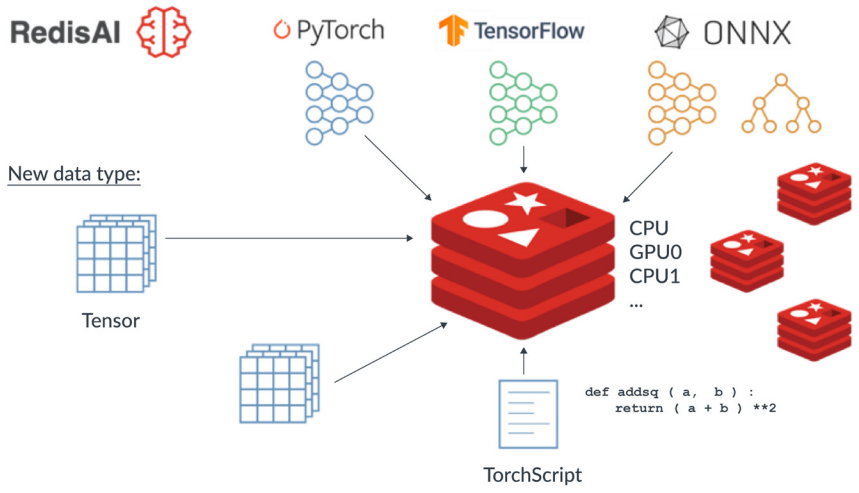
RedisAI, a solution for edge inference

Redis is a lightweight in-memory database used as a high-speed local key-value cache. Redis has a modular structure, which allows adding custom functionalities to run over stored data. RedisAI is a module written by RedisLabs for the purpose of running deep learning models over stored data. Running as a Redis module, RedisAI increases the throughput and reduces latency.

RedisAI on the edge can bring the following benefits:

- Latency reduction and throughput increase achieved through performing inference on the data directly from database's shared memory.
- Zero downtime on new model deployment as models can be updated transparently resulting in no operational downtime.
- Support for major backends, including Tensorflow, Tensorflow Lite, Pytorch and ONNXRuntime.
- With low footprint, RedisAI is highly scalable and increases throughput and available memory.

Figure 1. Redis AI components



Redis, RedisAI and Redis Streams work together

The RedisAI solution consists of several components:

- RedisAI: The module RedisAI is designed to apply inference on input tensors (multi-dimensional arrays storing data values) and return the result as output tensors stored in Redis database (A Tensor data structure is another data type that Redis supports). RedisAI allows models to be stored into Redis and run against input data.

- Redis Streams: Edge devices receive information as streams, a sequence of data values arriving from peripheral devices, sensors, or other systems. Data streams in Redis is handles by Redis Streams module, which allows streaming data to be ingested into the database and allows consumers to wait on the new data added to the stream to consume. Consumers can act as one or more consumer groups, which allows them to work in a distributed fashion for parallel processing when the rate of data ingestion is high.

- RedisGears: A component is required to glue Redis Streams and RedisAI, where it can read and filter streaming records, transform them into tensors, run inference on them, and store the results as another data stream or send it to an external storage. Another Redis module that is implemented to achieve such a dynamic programmable mechanism is RedisGears. It is a serverless engine, designed for batch and event driven data processing. RedisGears functions starts with a reader, which reads from a source and creates data records, and the next steps will perform custom operations on the incoming data. A function registers itself as event handler (like data stream events) so it runs on the new records whenever they are added to the stream.

- RedisGears allows users to connect RedisAI and Redis Streams in a programmatic way by providing Python and C APIs. The components of RedisGears are as the figure below:

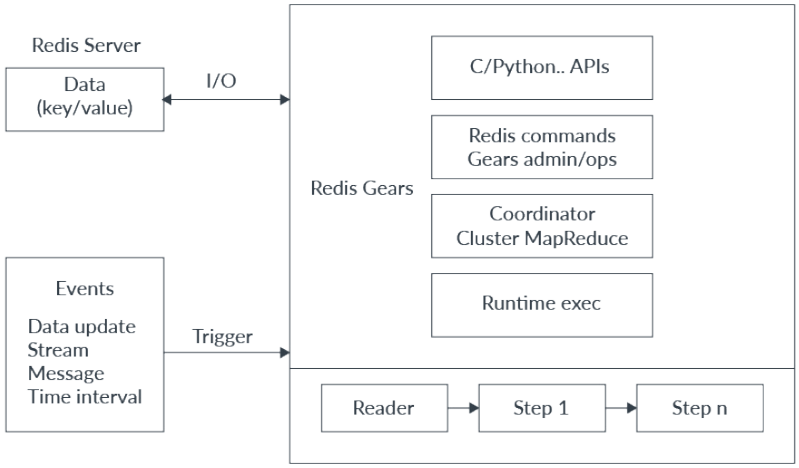


Figure 2. RedisGears components and internal design

RedisAI on Arm

RedisAI runs on Arm-based processors such as AWS Graviton2, Nvidia Jetson Nano, and Jetson TX2 out of the box. In addition, RedisAI can be containerized to run on Arm systems, which allows container orchestration platforms like Kubernetes to automate the deployment and updated of RedisAI clusters on edge devices or on the cloud.

RedisAI and Redis Streams in production

Orobix is one of Arm ecosystem partners that provides artificial intelligence solutions on Arm platforms to manufacturers and other businesses. It has developed an AI lifecycle and governance platform (named Invariant.ai) for managing artificial intelligence in critical contexts. By leveraging Redis Streams and Redis AI, it ensures observability of AI-governed processes and traceability of production models. Their solutions perform real-time analytics for anomaly detection, drift detection, similarity finding, and predictive maintenance.

Use case: Finger follower inspection in automotive industry

Orobix has designed an automated defect detection for finger follower parts produced by a motorcycle parts manufacturer. Finger followers are used in motorcycle engine to transfer the rotary motion from camshaft to valves. The system had to differentiate between actual defects and dust or small particles from industrial machinery. Before Orobix, the defect detection process was handled by human inspection. Orobix used RedisAI cluster for storing images and finding defects (anomalies) in production parts.

System components

The system design for automated classification of defective finger followers is shown in Figure 3. It uses

1. Arm processors for the rest of the components, specifically NVIDIA Jetson TX2 (VisionBox Daytona), with Quad-Core Arm Cortex-A57 CPU and 256-core NVIDIA Pascal GPU architecture. This allowed the customer to run ML/DL inferences on GPU cores, increasing the throughput dramatically.

2. Cameras with GigE Vision standard support (TeleDyne Dalsa or Basler).

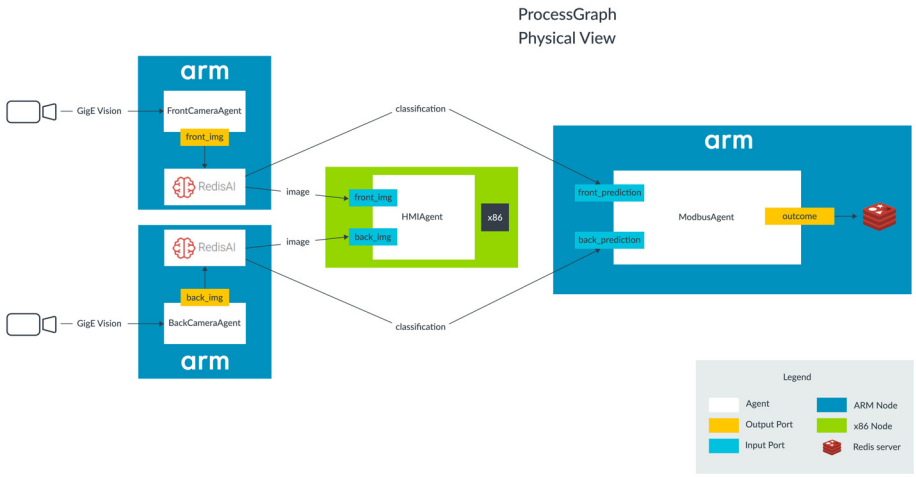
3. General purpose industrial PC for HMI (Human Machine Interface) agent.

Invariant.ai platform, which the system is design on top of it, is composed of agents connecting through communication channels. An agent is a software component implementing a certain functionality, and explicitly declaring the set of input and output ports. Each port is mapped to a Redis stream on a certain Redis instance and can either act as an input (or output) source of messages. Figure 3 shows a Process Graph, which is a concept used to describe the connections between agent ports and is used to configure how data flows across the different software components that implement the required function in a distributed manner. That provides

- full observability of the data flowing across the different software components
- ease of distribution of software components on different physical machines, decoupling the execution of software running on them

The process graph of the system is composed of 7 Invariant.ai agents in addition to RedisAI. Two agents perform the acquisition from two Industrial cameras (Teledyne DALSA) implementing the GigE Vision Protocol. One camera captures images of the front and the other one of the back of parts. Each image is written once to a Redis stream, and multiple agents can consume the same image. In fact, not only the HMI (Human Machine Interface) agent receives the produced images to show them as a preview to the end-user, but also the FrontClassification and BackClassification agents are receiving a copy of them. The Front and Back classification agents perform pre-processing of the received images and then leverage RedisAI for the inference (Deep Learning in this specific scenario). Each Classification agent produces an inference (defective/not defective), that is then processed by the Modbus agent. This last agent leverages the Modbus protocol to write to the PLC of the machine the final outcome.

Figure 3. Process graph of defect detection system



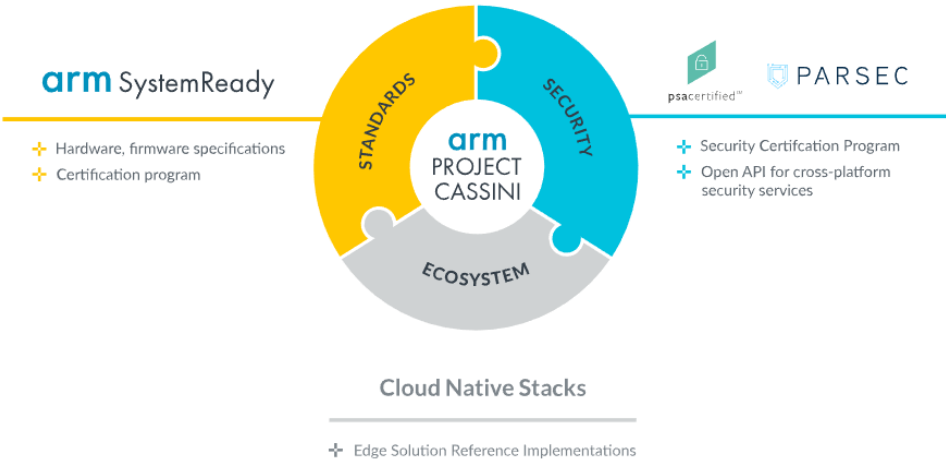
All the agents run in Docker containers. Each node capturing images from the cameras run two containers, one for the camera agent, and one running Redis. Similarly, the node running classification agents runs two containers, one for front image classification, and one for the back.

The modular system design helps in customizing hardware components based on resource requirements. For instance, larger models may require RedisAI to run on a machine with more memory, while the rest of the system remains intact. In addition, containerized architecture allows running software components on customized architecture with the least possible modification, and on different manufacturing environments.

Edge inference with Project Cassini

Project Cassini is a collaborative initiative covering standards, platform security, and reference implementations with cloud-native standards at edge. That allows users to run their software stack on edge devices and connect them to their cloud environment in a secure and scalable manner.

With more partners joining Project Cassini's edge ecosystem, users will have a broader choice on different platforms without experiencing difficulties. This is important for edge applications, where the same software may have to run on different hardware and firmware. Project Cassini allows frictionless orchestration of applications and connect them to the different cloud providers.



Cloud-native experience through Project Cassini helps edge analytics applications, where a single software may run on hundreds or thousands of devices, with simple and seamless management of application lifecycle, scalability, fault tolerance, and updates are necessary. Redis and RedisAI can run inside containers on Arm devices, hence benefit from cloud-native software stack for orchestration and management. In addition, SystemReady compliance allows applications to run on different Arm platforms with no modification required.

Take the Next Step

We invite partners to engage on Project Cassini and explore the applicability of the RedisAI for their edge inference solutions. To get involved, contact us at project-cassini@arm.com or visit us at arm.com/project-cassini.