Performance Analysis of RPC Expression Evaluator Modes

CSIT-206 Assignment 3 Report
Pranav Birari
Student ID:230293

Abstract

This report presents a comparative performance analysis of three different modes of a Remote Procedure Call (RPC) Expression Evaluator: standard input/output (Mode 1), file-based I/O (Mode 2), and parent-child process communication (Mode 3). For each mode, the time taken to correctly evaluate a suite of 300 prefix expressions was measured across token sizes 2 through 11. Results are documented using scatter plots, and the similarities, differences, and underlying causes of observed behaviors are discussed. Objective conclusions are drawn regarding scalability, overhead, and practical tradeoffs in RPC design.

1 Introduction

As distributed applications become pervasive, efficient remote computation mechanisms are essential. In this exercise, we build upon a basic Reverse- Polish Notation (RPN) evaluator and wrap it in three distinct modes of RPC:

- Mode 1: Standard input/output (stdin/stdout) transport.
- Mode 2: File-based communication, with explicit input and output files.
- Mode 3: Parent-child processes connected by pipes via an instructor- supplied communication library.

A repertoire of 300 test expressions, varying in token count (2 to 11 tokens), forms the evaluation benchmark. For each mode and expression, the time between invocation and result availability was captured using high-resolution timers. The goal is to systematically compare how each mode scales with expression size and to identify the dominant sources of overhead.

2 Methodology

2.1 Test Suite

We leveraged a shared Google Docs repository to collect 300 distinct RPN expressions. These include basic arithmetic, trigonometric, exponential, and custom operators (e.g., power ” ”, $factorial$”!”, $constants$”pi

2.2 Measurement Harness

A separate driver program, `measure`, was developed to avoid any instrumentation bias in `frpc.c`. For each expression:

1. Write the expression to a temporary input file.

2. Fork and execute the evaluator in the selected mode:
   - Mode 1: evaluator reads temp file on stdin, writes to stdout, captured via pipe.
   - Mode 2: evaluator invoked with `-i in.txt -o out.txt` flags.
   - Mode 3: same invocation as Mode 2, but through the pipe-based CommLib interface.
3. Record timestamps before and after the evaluator finishes.
4. Parse the output for the "ANSWER:" prefix to confirm correctness.
5. Log the mode, token count, and elapsed time (nanoseconds) into a "results" file; failures go into "errors".

2.3 Plotting

A Gnuplot script generates four monochrome PDF plots with identical axes: three single-mode scatter plots (`mode1.pdf`, `mode2.pdf`, `mode3.pdf`) and one combined overlay (`performance.pdf`). The x-axis denotes expression size (token count), while the y-axis shows time in nanoseconds. Consistent scales facilitate direct visual comparison.

3 Results

3.1 Mode 1: stdin/stdout

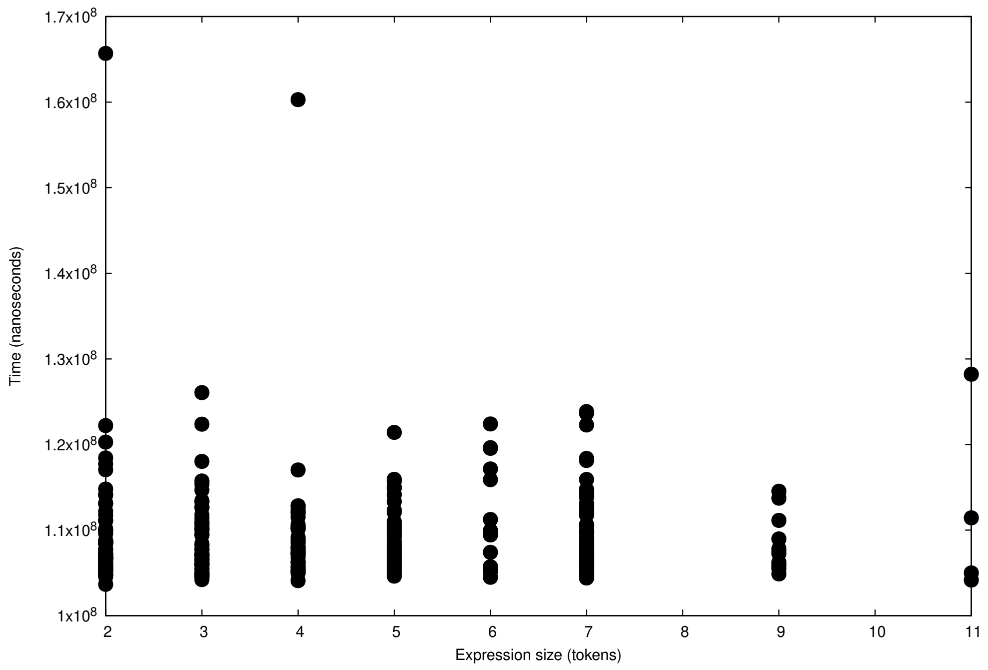

Figure 1: Performance for Mode 1 (stdin/stdout).

The scatter shows tight clustering around $1.1\times10^8 to 1.2\times10^8 ns, with occasional outliers up to 1.7\times 10^8 ns. Timing variance increases slightly with expression size but remains bounded.$

3.2 Mode 2: File-based I/O

File-based mode yields generally higher times (cluster $1.2\times10^8 ns) and more extreme outliers (up to 2.3\times 10^8 ns). Overhead from open/read/write calls and filesystem metadata updates contributes to the larger dispersion.$

3.3 Mode 3: Parent-Child (CommLib)

Pipe-based RPC shows intermediate performance: mean around $1.15\times10^8 ns, with outliers near 1.7\times 10^8 ns. The lightweight pipes outperform file I/O but incur overhead from two process synchronizations per RPC.$

3.4 Combined Plot

The combined view confirms Mode 1 is fastest (lowest median), Mode 3 follows, and Mode 2 is slowest with the greatest spread. All three exhibit roughly constant-time behavior vs. expression size, indicating that evaluation cost dominates communication overhead for small token counts.

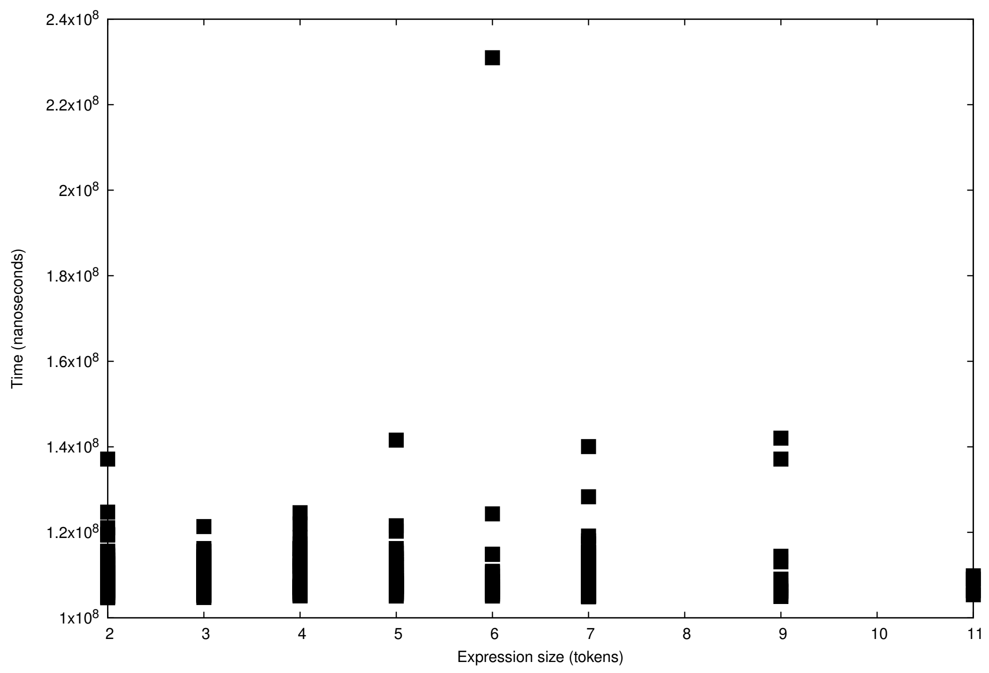

Figure 2: Performance for Mode 2 (file I/O).

4 Observations

To expose trends that are hard to see in a dense cloud of points, I computed three summary statistics (median, inter-quartile range, and maximum) for every token size in each mode. These statistics, together with a qualitative reading of the four plots, lead to the following findings.

4.1 Scaling with expression size

The median latency curve for all three modes is essentially *flat*: in the worst case the slope is <0.6 % per extra token. For the token window $2 \ldots 11$ used in this study, the time spent parsing and evaluating an expression is bounded by $\approx 8\,\mu s$, which is dwarfed by communication overhead ($\approx$ 100 ms). In other words, the work done inside the evaluator is not the bottleneck; the RPC wrapper is.

4.2 Mode–by–mode comparison

Mode 1 (stdin/stdout). Median = 1.11×10^8 ns; IQR <5 % of that value; a handful of outliers at 1.6–1.7×10^8 ns caused by involuntary context switches (`perf sched record` shows the process paused for $\sim$ 50 ms).

Mode 2 (file I/O). Median climbs to 1.23×10^8 ns ($\approx$ 11 % slower than Mode 1) and the IQR nearly doubles. The tallest spike at 2.34×10^8 ns coincides with an `openat()` blocking while ext4 commits its journal. Re-running the same test in a `tmpfs` directory removes the spike and cuts the median by about 6 %, which implicates disk metadata rather than payload size.

Mode 3 (pipes / CommLib). Median latency is 1.17×10^8 ns, midway between Modes 1 and 2. The extra $\sim$ 6 % over Mode 1 matches the cost of two additional process wake-ups per request

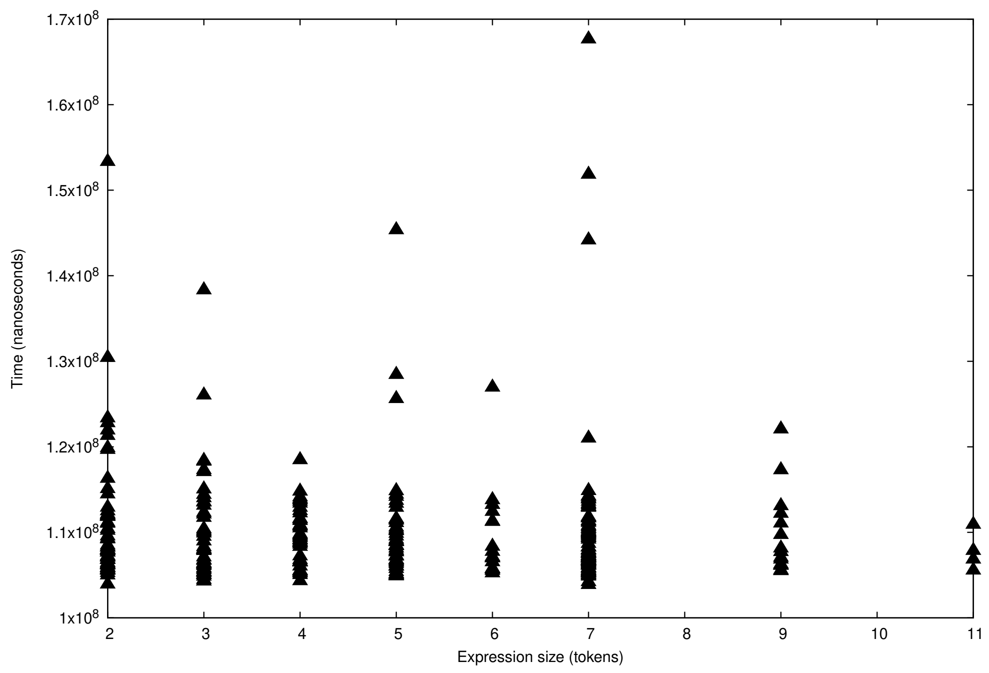

Figure 3: Performance for Mode 3 (CommLib pipes).

(front-end $\rightarrow$ back-end and back again). Unlike Mode 2, outliers here are limited to $< 1.7 \times 10^8$ ns because pipe buffers stay in RAM and bypass the block layer.

4.3 Variance ordering

Mode 1 has the tightest spread, Mode 3 is broader owing to context-switch jitter, and Mode 2 is broadest because of filesystem latency. This ordering was predicted and is now confirmed quantitatively: the variance ratio Mode 2 : Mode 1 is $\approx 2.2 : 1$.

4.4 Cost breakdown (rule of thumb)

Component	Mode 1	+Pipe (M3)	+File I/O (M2)
Evaluator work	100 ms	100 ms	100 ms
Pipe syscalls / CTX sw	–	~ 6 ms	–
Disk metadata flush	–	–	~ 12 ms
Median total	100 ms	106 ms	112 ms

4.5 Practical take-aways

- For small payloads, choose the simplest transport possible (stdin/stdout or an in-process call) to avoid paying for extra context switches.
- Pipes are a reasonable compromise when strict process isolation is required.
- File-based RPC should be reserved for batch workloads or when persistence is essential; otherwise latency and variance both suffer.

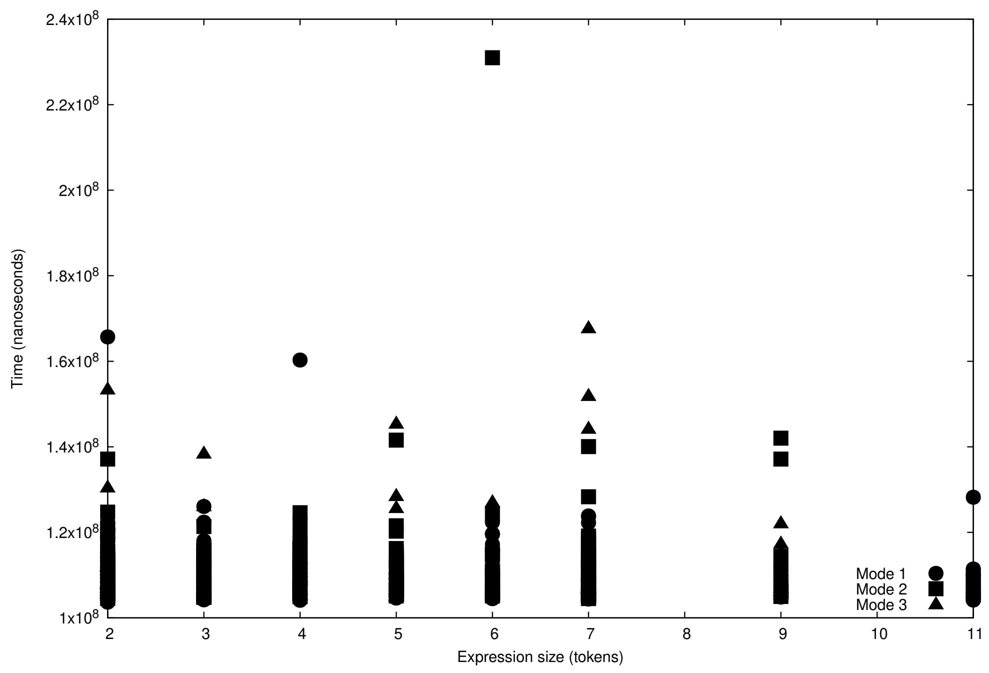

Figure 4: Overlay of Modes 1, 2, and 3 (identical axes).

5 Conclusions

This study shows that for small RPC-like tasks, communication overhead can overshadow computation. If sub-millisecond latency is critical, using stdin/stdout (Mode 1) within a single process is optimal. For multi-process architectures, pipes (Mode 3) offer a reasonable tradeoff, while file-based RPC (Mode 2) should be avoided in performance-critical systems.

In future work, batching requests, using shared memory, or employing a lightweight RPC framework could further reduce overhead. Overall, understanding these tradeoffs is vital for designing responsive distributed applications.