西安郵電大學

毕业设计（论文）

题　　目：基于 eBPF 的程序访存性能监测设计及可视化系统与实现

学　　院：计算机学院

专　　业：网络工程

班　　级：2101

学　　号：04212032

学生姓名：高鹏

导师姓名：王亚刚　职称：副教授

起止时间：2024 年 11 月 20 日至 2025 年 6 月 6 日

年　　月　　日

毕业设计（论文）承诺书

本人所提交的毕业设计（论文）《Bitmessage 协议库的设计与实现》是本人在指导教师指导下独立研究、写作的成果，毕业设计（论文）中所引用他人的文献、数据、图件、资料均已明确标注；对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式注明并表示感谢。

本人深知本承诺书的法律责任，违规后果由本人承担。

论文作者签名：_____ 日 期：_____

关于毕业设计（论文）使用授权的声明

本人在导师指导下所完成的论文及相关的职务作品，知识产权归属西安邮电大学。本人完全了解西安邮电大学有关保存、使用毕业设计（论文）的规定，同意学校保存或向国家有关部门或机构送交论文的纸质版和电子版，允许论文被查阅和借阅；本人授权西安邮电大学可以将本毕业设计（论文）的全部或部分内容编入有关数据库进行检索，可以采用任何复制手段保存和汇编本毕业设计（论文）。本人离校后发表、使用毕业设计（论文）或与该毕业设计（论文）直接相关的学术论文或成果时，第一署名单位仍然为西安邮电大学。

本毕业设计（论文）研究内容：

□可以公开

□不宜公开，已办理保密申请，解密后适用本授权书。

（请在以上选项内选择其中一项打“ √ ”）

论文作者签名：_____ 导师签名：_____

日 期：_____ 日 期：_____

西安邮电大学本科毕业设计(论文)选题审批表

申 报 人	王亚刚	职称	副教授	学院	计算机学院
题目名称	基于 eBPF 的程序访存性能监测及可视化系统设计与实现				
题目来源	☑教师科研课题 ☐教师专业实践 ☐其他				
题目类型	☐艺术作品 ☐硬件设计 ☐软件设计 ☑论文				
题目分类	☑工程实践 ☐社会调查 ☐实习 ☐实验 ☐其他				
题目简述	在国产芯片的软件编译和链接的相关科研项目中，了解程序运行时的访存性能指标，尤其是内存访问过程中各级缓存（Cache）的命中率、缺失率，缺页率、访问时间等，是进行程序存储布局优化的关键因素和系统性能评价的重要指标。目前的主要性能检测工具，例如 perf，规模比较庞大，运行代价高，而且存在一定的误差。本项目基于 eBPF 框架，在 Linux 内核中建立一定的存储访问观测点，通过挂载相应的信息采集模块，直接从 Linux 内核中获取相关的访存信息，并对这些信息进行整理，最终形成可视化结果，为研究不同程序存储布局的可执行程序的性能评估提供依据。				
对学生知识与能力要求	1.熟悉 Linux 操作系统 2.熟悉 eBPF 框架，能够利用 eBPF 程序进行内核监控程序的开发 3.了解 Linux 系统中存储访问模块的设计与实现 4.熟悉 WEB 开发，将最终的观测数据进行可视化展示。				
具体任务以及预期目标	本课题的具体任务包括： 在 Linux 系统上开发基于 eBPF 的程序内存访问的监控程序，观测程序加载和运行过程中存储访问的主要过程，提取重要的内存访问信息，关注 Cache 和缺页处理，并对这些信息进行采集、整理，最后将这些数据基于 web 进行展示，实现程序运行过程中存储访问性能的可视化。 具体的预期目标及成果形式包括： 1.程序运行过程中存储访问的基本过程分析，给出分析文档 1 份 2.使用 eBPF 框架，开发一个系统，实现程序运行过程中存储访问重要信息的提取。 3.开发 web 前后端，实现信息的可视化。				

<table>
<tr><td>时间
进度</td><td colspan="3">2024 年 11 月 25 日-11 月 24 日：完成毕业设计选题
2024 年 11 月 25 日-2025 年 1 月 10 日：提交开题报告，前期检查
2025 年 1 月 11 日-3 月 29 日：完成环境搭建，并完成前后台接口设计，中期检查
2025 年 3 月 30 日-5 月 17 日：完成设计实现代码，进行代码验收
2025 年 4 月 1 日-5 月 25 日：撰写毕业论文
2025 年 5 月 26 日 6 月 1 日：完善毕业论文，进行论文答辩。</td></tr>
<tr><td>专业负责人审核意见</td><td colspan="3">签字：　　年　月　日</td></tr>
<tr><td>系（教研室）主任签字</td><td>年　月　日</td><td>主管院长签字</td><td>年　月　日</td></tr>
</table>

西安邮电大学本科毕业设计（论文）开题报告

学生姓名	高鹏	学号	04212032	专业班级	网络工程 2101
指导教师	王亚刚	题目	基于 eBPF 的程序访存性能监测及可视化系统设计与实现		

选题目的（为什么选该课题）

在国产芯片的软件编译和链接的相关科研项目中，需要了解程序运行时的访存性能指标，尤其是内存访问过程中各级缓存（Cache）的命中率、缺失率，缺页率、访问时间等。缓存命中（Cache Hit）：当 CPU 需要访问某个数据时，如果该数据已经存储在缓存中，就称为缓存命中。缓存命中意味着可以快速获取数据，减少延迟；缓存缺失（Cache Miss）：当 CPU 需要访问某个数据时，如果该数据不在缓存中，就称为缓存缺失。缓存缺失意味着需要从更低一级的缓存或主存中获取数据，增加访问延迟；缺页（Page Fault）：当 CPU 需要访问某个虚拟内存地址时，如果该地址对应的页面不在物理内存中，就会发生缺页。操作系统需要将页面从磁盘加载到物理内存中，然后重新尝试访问该页面。缺页通常会显著影响性能；访问时间（Access Time）：访问时间是内存访问请求的延迟，包括从发出请求到数据返回的所有时间。访问时间受到缓存命中率、缓存层级、内存带宽等因素的影响。以上这些指标是进行程序存储布局优化的关键因素和系统性能评价的重要指标。

目前对访存性能指标进行监测大多采用传统的 perf 监测工具。perf 是 Linux 内核自带的性能分析工具，主要用于收集和分析各种硬件和软件的性能事件。perf 可以通过定期采样或者基于事件触发来收集性能数据。但是它存在一些弊端：perf 的运行开销与采样频率密切相关。如果设置了较高的采样频率（例如每毫秒采样一次），则会占用更多的 CPU 资源，从而对被监测程序的性能产生较大的影响；由于 perf 依赖于定期采样，可能会错过一些短暂但重要的性能事件，导致采样误差；采集到的性能数据需要存储和处理，在高负载情况下，这些操作可能会导致额外的系统开销。

本项目采用的 eBPF 是可以在内核虚拟机中运行的程序，可以执行 eBPF 字节码，可以在不重启内核，不加载内核模块的情况下动态安全的扩展内核功能；通过 kprobe/ uprobe/ tracepoints 等动态监视、修改系统状态，可以提取细粒度的安全可观察性数据。eBPF 的运行开销通常比 perf 更低，eBPF 程序是在内核中执行的，避免了频繁的用户态和内核态切换带来的开销；eBPF 可以动态插入到内核的各个探测点，可以精确地监控特定内核事件或用户态程序的行为；可以精确控制监测的事件和触发条件减少不必要的数据采集和处理。

将 perf 与 eBPF 进行对比，更能体现出 eBPF 的优势：（1）采样频率：perf 依赖于定期采样，高频采样会带来显著的性能开销。eBPF 可以在特定事件发生时触发，避免了高频采样的开销；（2）数据处理：perf 会将采集的数据存储到用户空间，可能导致较高的 I/O 开销。eBPF 可以在内核态处理数据，减少了用户态和内核态之间的数据传输；（3）灵活性和精确度：perf 的采样方法可能会错过一些短暂事件或引入系统噪声。eBPF 可以精确地插入到特定代码路径或内核事件中，提供更高的监测精度。

总而言之，本项目基于 eBPF 框架，在 Linux 内核中建立一定的存储访问观测点，通过挂载相应的信息采集模块，直接从 Linux 内核中获取相关的访存信息，并对这些信息进行整理，最终形成可视化结果，更加直观、高效、准确，为研究不同程序存储布局的可执行程序的性能评估提供依据。对于操作系统开发者和架构师来说，通过对不同程序内存访问模式的分析，可以为系统设计和硬件配置提供参考。

参考文献

[1] ALAM M, HAMDI M, ZEIDAN H. A novel approach for memory performance analysis using eBPF[C]// 2021 12th International Conference on Computing, Communication and Networking Technologies (ICCCNT). IEEE, 2021: 1-6.
[2]D. Borkmann and A. Starovoitov, “eBPF and Linux Kernel Security: A Comprehensive Analysis,” in Proceedings of the USENIX Annual Technical Conference (USENIX ATC), 2021, pp. 123–138.
[3]B. Gregg, BPF Performance Tools: Linux System and Application Observability. Boston, MA, USA: Addison-Wesley Professional, 2019.
[4]T. Xu et al., “MemAxes: A Tool for Visualizing Memory Access Patterns,” IEEE Transactions on Visualization and Computer Graphics, vol. 26, no. 1, pp. 860–870, Jan. 2020.
[5]M. Canini et al., “A High-Speed Network Monitoring Framework with eBPF,” in Proceedings of the ACM SIGCOMM Conference, 2016, pp. 123–135.
[6]M. Seltzer et al., “Kernel Instrumentation with eBPF: A Case Study in File System Monitoring,” in Proceedings of the USENIX Annual Technical Conference (USENIX ATC), 2017, pp. 45–58.
[7]S. Kaxiras and A. Ros, “Cache Hierarchy-Aware Memory Access Analysis,” ACM Transactions on Architecture and Code Optimization, vol. 17, no. 3, pp. 1–25, Sep. 2020.
[8]M. Bostock et al., “D3: Data-Driven Documents,” IEEE Transactions on Visualization and Computer Graphics, vol. 17, no. 12, pp. 2301–2309, Dec. 2011.
[9]A. Starovoitov, “BPF: A New Type of Software,” in Proceedings of the Linux Plumbers Conference, 2018, pp. 1–10.
[10]A. Bhattacharjee et al., “Efficient Page Fault Handling in Modern Operating Systems,” ACM SIGOPS Operating Systems Review, vol. 47, no. 2, pp. 34–49, 2013.

前期基础（已学课程、掌握的工具、资料积累、软硬件条件等）

1.已学课程

《数据结构》、《计算机组成原理》、《操作系统》、《计算机网络》、《软件工程》、《数据库原理及运用》、 《Web 开发技术》

2.掌握的工具

JavaScript(前端开发语言) 、C 编程语言、libpf(eBPF 开发框架)、Visual Studio Code 开发工具、 Git（分布式版本控制系统）等

3.实践积累

eBPF 项目开发经历，实习中了解内存的性能优化手段

4. 软硬件条件

Ubuntu 22.04（Linux 6.15）操作系统，Visual Studio Code 1.85.0 版本

要研究和解决的问题（做什么）

（1）eBPF 环境搭建：首先需要搭建 eBPF 开发环境，并深入分析选用 bcc（BPF Compiler Collection）或 libpf 框架的优劣势。考虑到不同 Linux 内核版本的兼容性问题，需要评估各个内核版本对 bpfhelper 函数的支持情况，确保代码的可移植性和跨版本运行的稳定性。

（2）选取钩子函数：在内核中合适的位置插入钩子函数是实现存储访问监控的关键。需要分析使用 tracepoint 或 kprobe 来挂载合适的钩子函数。为此，需要深入阅读和分析 Linux 内核源码，确定最佳的插入点，以便准确捕获所需的访存信息。

（3）测试监控性能和数据的准确性：为了确保系统的实用性，需要对监控机制的性能和数据的准确性进行严格测试。包括评估监控工具对被测程序运行性能的影响，验证数据采集的精确度，以及确保在高负载情况下程序的稳定性和可靠性。

（4）开发 Web 前后端，实现可视化：为了让研究结果和监控数据更直观易读，需要开发一个 Web 前后端系统，实现数据的可视化展示。前端应具备友好的用户界面，能够动态展示内存访问模式和性能分析结果；后端则负责数据处理和分析，将 eBPF 收集到的信息进行整理并传递给前端展示

工作思路和方案（怎么做）

（1）框架选用：

选用 libpf 框架进行 eBPF 程序开发，利用 BPF_CORE（BPF Compile Once - Run Everywhere）技术确保代码在不同内核版本上的兼容性。通过 BPF_CORE，能够在编译时生成适用于多个内核版本的 eBPF 代码，提高代码的可移植性和稳定性。

（2）需求分析：

分析程序访存过程需要调用的内核函数，并且选用合适的 hook 方式来捕捉所需事件。Tracepoints：内核中预定义的跟踪点，适用于捕捉特定事件，具有较高的稳定性。具体选择如下：

kmem:mm_page_alloc 和 kmem:mm_page_free：用于跟踪内存页面的分配和释放情况，从而分析内存使用情况。

sched:sched_process_exec 和 sched:sched_process_exit：用于跟踪进程的启动和退出事件，分析内存使用与进程生命周期之间的关系。

Kprobes：动态插桩技术，可以挂载到几乎任何内核函数上，灵活性高。具体选择如下：

do_page_fault：用于捕捉页面错误事件，从而分析缺页率。

handle_mm_fault：用于进一步细分页面错误类型，分析不同类型页面错误的分布情况。

(3) 数据交互和传输：

使用 eBPF 的 MAP 机制进行数据存储和传输。为了提高数据传输性能，采用 ringbuf 或 perfbuf 技术。MAP：用于存储和共享数据，保证数据在内核空间和用户空间之间的交互。ringbuf 或 perfbuf：用于高效的数据传输，减少数据传输的延迟和开销。

(4) 测试测试方案包括以下步骤：

功能测试：验证每个 hook 点是否正确捕获目标事件。检查数据传输的正确性和完整性。

性能测试：测试监控工具对被测程序运行性能的影响，包括 CPU 占用率和内存使用情况。在不同负载条件下，评估监控工具的性能和稳定性。

准确性测试：对比监控数据和实际系统行为，验证数据的准确性。使用多种测试场景，确保监控数据在各种情况下都能准确反映系统状态。

(5) 可视化信息呈现方式：开发一个 Web 前后端系统，实现数据的可视化展示。
前端开发：使用现代 Web 技术，如 React 或 Vue.js，构建友好的用户界面。实现动态数据展示和交互功能，包括图表、表格和实时更新。提供多种视图选项，便于用户按需查看内存使用情况、进程生命周期、页面错误统计等信息。
后端开发：使用 Node.js 或 Python 等语言，构建后端数据处理和接口服务。实现与 eBPF 程序的数据交互，接收并处理内核传递的数据。将处理后的数据传递给前端，支持实时更新和历史数据查询。
数据可视化：使用图表库（如 D3.js 或 Chart.js）实现数据的可视化展示。通过折线图、柱状图、饼图等多种形式，直观呈现内存使用、页面分配和释放、页面错误等信息。提供数据筛选和过滤功能，帮助用户快速找到所需信息。

指导教师意见

签字　　年　月　日

西安邮电大学毕业设计（论文)成绩评定表（理工）

<table>
<tr><td>学生姓名</td><td>高鹏</td><td>性别</td><td>男</td><td>学号</td><td>04212032</td><td>专业班级</td><td>网络工程
网络 2101</td></tr>
<tr><td>课题名称</td><td colspan="7">基于 eBPF 的程序访存性能监测及可视化系统设计与实现</td></tr>
<tr><td rowspan="3">指导教师意见</td><td>支撑指标点/赋分</td><td>3-2/20</td><td>4-2/20</td><td>5-3/10</td><td>7-2/10</td><td>8-2/10</td><td>11-2/10</td><td>12-2/20</td><td>合计</td></tr>
<tr><td>得分</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="8">指导教师(签字)：　　年　月　日</td></tr>
<tr><td rowspan="3">评阅教师意见</td><td>支撑指标点/赋分</td><td></td><td></td><td></td><td></td><td></td><td></td><td>合计</td></tr>
<tr><td>得分</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="8">评阅教师(签字)：　　年　月　日</td></tr>
<tr><td rowspan="3">验收小组意见</td><td>支撑指标点/赋分</td><td></td><td></td><td></td><td></td><td>合计</td></tr>
<tr><td>得分</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="6">验收小组组长(签字)：　　年　月　日</td></tr>
<tr><td rowspan="3">答辩小组意见</td><td>支撑指标点/赋分</td><td></td><td></td><td></td><td></td><td>合计</td></tr>
<tr><td>得分</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="6">答辩小组组长(签字)：　　年　月　日</td></tr>
<tr><td rowspan="3">学生总评成绩</td><td>评分比例</td><td>指导教师(20%)</td><td>评阅教师(30%)</td><td>验收小组(20%)</td><td>答辩小组(30%)</td><td>合计</td></tr>
<tr><td>评分</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="4">毕业论文(设计)最终等级制成绩（优秀、良好、中等、及格、不及格）</td><td colspan="2"></td></tr>
<tr><td>答辩委员会意见</td><td colspan="7">学院答辩委员会主任(签字、学院盖章)：　　年　月　日</td></tr>
</table>

摘　要

eBPF（Extended Berkeley Packet Filter）[1]技术近年来在 Linux 系统中迅速发展，为系统观测和性能分析提供了新的可能性。本文工具利用 eBPF 技术，针对国产芯片软件编译和链接研究中对访存性能精确监测的需求，为系统优化、程序性能调优以及计算机体系结构研究提供了有价值的工具支持。

本文的主要内容包括：

1.为了精确分析和可视化程序内存访问行为，设计并实现了一种基于 eBPF 的程序运行时访存性能监测工具——eMemGazer。本工具利用 eBPF 技术直接在内核层面捕获并分析内存访问事件，包括各级缓存（L1、LLC）命中率、缺失率，TLB 性能，缺页异常及 CPU 指令执行效率等关键指标，并基于 WEB 实现了指标数据的可视化。

2.为了验证 eMemGazer 工具的正确性及性能，选取 Linux 内核自带的 perf 工具作为基准，分别在顺序访问、随机访问和固定步长访问三种典型内存访问模式进行测试。实验中 eMemGazer 获取的性能指标与 perf 工具的结果高度一致，验证了 eMemGazer 在功能正确的基础上，同时在系统开销、实时性和精度方面表现出优势。

关键词：eBPF；内存性能监测；缓存命中率；性能优化；

Abstract

eBPF (Extended Berkeley Packet Filter) technology has rapidly evolved in Linux systems in recent years, providing new possibilities for system observation and performance analysis. This paper presents a tool that leverages eBPF technology to address the precise memory access performance monitoring requirements in software compilation and linking research for domestic chips, offering valuable tool support for system optimization, program performance tuning, and computer architecture research.

The main contributions of this paper include:

1. To accurately analyze and visualize program memory access behavior, we designed and implemented an eBPF-based runtime memory performance monitoring tool called eMemGazer. This tool utilizes eBPF technology to directly capture and analyze memory access events at the kernel level, including key metrics such as cache hit rates and miss rates at various levels (L1, LLC), TLB performance, page fault exceptions, and CPU instruction execution efficiency. The tool also implements web-based visualization of metric data.

2. To validate the correctness and performance of the eMemGazer tool, we selected the Linux kernel's built-in perf tool as a benchmark and conducted tests under three typical memory access patterns: sequential access, random access, and fixed-stride access. The experimental results show that the performance metrics obtained by eMemGazer are highly consistent with those from the perf tool, validating that eMemGazer not only maintains functional correctness but also demonstrates advantages in terms of system overhead, real-time performance, and precision.

Keywords: eBPF; memory performance monitoring; cache hit rate; performance optimization;

目 录

第 1 章 绪论

1.1 研究背景和意义

在现代计算机系统中，内存访问性能已成为程序执行效率的关键瓶颈。计算能力的飞速提升使得 CPU 与内存之间的速度差距不断扩大，这一现象被称为"内存墙"（Memory Wall）问题。为了缓解这一问题，现代计算机体系结构引入了复杂的多级缓存层次结构，包括 L1、L2 和 LLC 缓存，以及 TLB 等地址转换缓存。程序的执行效率在很大程度上取决于对这些缓存资源的有效利用。

在国产芯片及其软件生态构建过程中，深入理解并优化程序的内存访问行为具有重要意义。国产芯片通常具有独特的存储层次结构和缓存特性，针对这些特性优化程序的存储布局和访问模式能够显著提升性能。因此，实时、精确地监测程序运行时的访存性能指标，尤其是内存访问过程中各级缓存的命中率、缺失率，缺页率、访问时间等，成为程序优化的关键前提[5]。

传统的性能分析工具如 Linux perf 虽然功能强大，但由于依赖周期性采样，往往会引入较大的性能开销，并且可能错过一些短暂但重要的性能事件。而新兴的 eBPF 技术为内核级性能监测提供了强大而高效的机制，可以在几乎不影响被监测程序执行的前提下，精确捕获关键的性能事件。

本研究的意义在于：（1）提供了一种低开销、高精度的内存访问性能监测方法，为国产芯片软件优化提供重要的技术支持；（2）通过精确的性能数据采集和直观的可视化呈现，帮助开发者快速识别程序中的内存访问瓶颈；（3）为计算机体系结构和操作系统研究提供了实用的实验工具，可用于验证和评估新的内存子系统优化算法。

1.2 国内外研究现状

1.2.1 传统的程序运行时的访存性能指标监测

传统的程序运行时访存性能监测主要采用两种方式：软件模拟和硬件计数器。软件模拟方法如 Valgrind 的 Cachegrind 工具，通过模拟 CPU 和内存缓存的行为来跟踪程序的内存访问模式。这种方法虽然能提供详细的缓存访问统计信息，但会导致程序执行速度显著降低，通常为原速度的 10-50 倍，因此不适用于生产环境或大型应用程序的性能分析。

另一种传统方法是使用特定的代码插桩技术，如 PIN 工具或 LLVM Pass，在程序的关键点插入性能计数代码。这种方法可以提供精确的性能数据，但需要重新编译程序或进行复杂的二进制插桩，且插桩本身会改变程序的执行特性，引入较大的性能开销。

1.2.2 基于硬件性能计数器(PMC)的工具

现代处理器普遍集成了硬件性能监控单元，提供了一系列可编程的性能计数器，用于记录各种微架构事件，如缓存命中、缺失、分支预测失败等。基于 PMU 的工具，以 Linux perf 为代表，通过访问这些硬件计数器来分析程序性能。

Linux perf 是目前应用最广泛的性能分析工具之一，能够对 CPU 性能计数器、tracepoint 事件等进行采样。perf 的工作原理是周期性地中断 CPU，记录当前的性能计数器值和程序状态。这种基于采样的方法存在局限性：采样精度与开销的权衡，提高采样频率可以获得更准确的数据，但会增加系统开销，降低采样频率则可能错过重要事件；系统干扰，perf 采样过程中需要频繁地在用户态和内核态之间切换，可能会干扰被监测程序的正常执行，数据处理开销，收集的大量采样数据需要存储和后处理，在高负载情况下可能导致额外的系统压力。

1.2.3 基于 eBPF 的新兴监测技术

eBPF（Extended Berkeley Packet Filter）技术近年来在 Linux 系统中迅速发展，为系统观测和性能分析提供了新的可能性。eBPF 允许用户空间程序将自定义代码安全地注入到内核中的关键位置，无需重启系统或加载内核模块。相比传统方法，eBPF 具有以下优势：低开销，eBPF 程序直接在内核中执行，避免了频繁的用户态和内核态切换，大幅降低性能监测的开销[7]；精确事件触发，eBPF 可以在特定内核事件发生时精确触发，而不是依赖定期采样，从而提高了监测的精度[2]。

现有的 eBPF 工具在内存性能分析方面仍存在整合度不高、可视化能力有限等不足。

1.3 论文主要工作

本论文的主要工作包括以下几个方面：

设计并实现了基于 eBPF 的综合内存性能监测系统 eMemGazer：该系统能够实时监控程序运行过程中的各级缓存访问情况、TLB 性能、缺页异常以及 CPU 指令执行效率等关键性能指标，为程序内存性能优化提供全面的数据支持。

构建了直观的性能数据可视化系统：设计实现了交互式可视化界面，通过多维度图表展示内存访问性能数据，帮助用户快速识别性能瓶颈。

设计了自动化测试框架：开发了专用的内存访问模式测试程序和对比测试脚本，实现了 eMemGazer 与 Linux perf 工具的自动化对比测试，验证了系统的准确性和有效性。

1.4 文章结构

本论文的结构安排如下：

第一章 绪论：介绍研究背景和意义，分析国内外研究现状，概述论文的主要工作和结构安排。

第二章 背景知识和技术：阐述与本研究相关的基础知识，包括计算机内存层次结构、缓存工作原理、Linux 内核体系结构、eBPF 技术基础及其在性能监测中的应用。

第三章 eMemGazer 系统设计与原理：深入分析 eMemGazer 的核心原理，包括 eBPF 监控机制、性能事件捕获方法、eBPF 程序的生命周期管理、CO-RE 技术应用以及与传统 perf 工具的原理对比。

第四章 eMemGazer 系统实现：详细阐述系统的设计与实现，包括系统架构、内核态监测组件、用户态数据处理组件、可视化系统设计以及系统各模块的交互机制和实现难点解决方案。

第五章 系统测试与评估：描述测试环境配置，展示不同内存访问模式下的性能测试结果，分析 eMemGazer 与 perf 工具的对比数据，评估系统的准确性、低开销特性和可用性。

第六章 结论与展望：总结研究成果，分析系统的创新点和局限性，展望未来的研究方向和改进可能。

第二章 背景知识和技术

本章全面阐述了与内存性能监测相关的背景知识和技术基础。Linux 平台上现有的性能监控工具以及 eBPF 技术。

2.1 计算机内存层次结构

现代计算机系统采用层次化的内存架构，从 CPU 寄存器到辅助存储设备，形成了一个金字塔结构。这种设计旨在平衡速度、容量和成本之间的矛盾。在这个层次结构中，从上到下依次是：CPU 寄存器、多级缓存（L1、L2、L3/LLC）、主内存（RAM）、固态硬盘（SSD）或机械硬盘（HDD）等辅助存储设备。

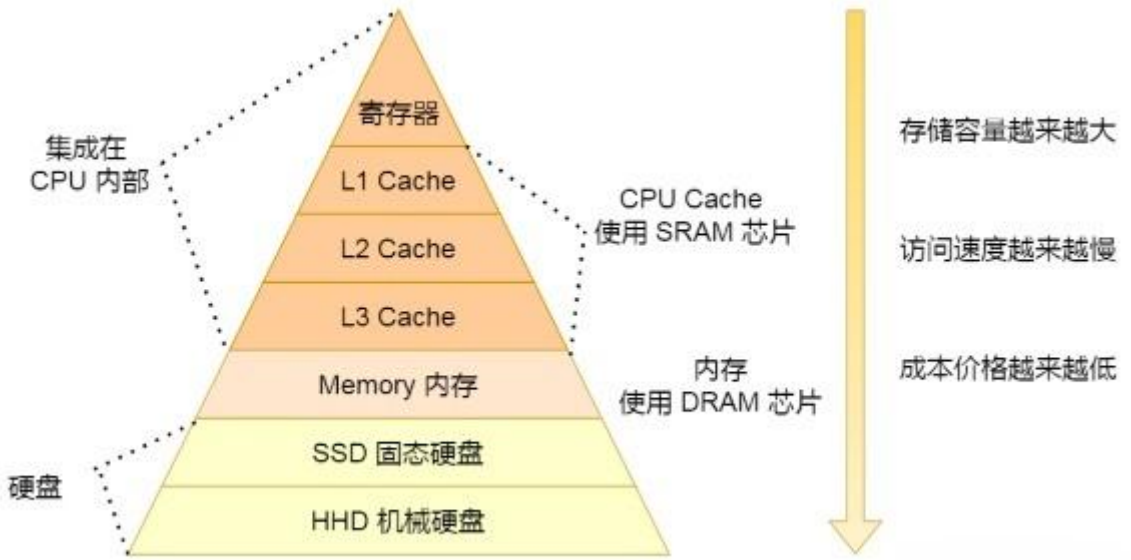


图 2.1 计算机存储层次结构图

L2 缓存容量在几百 KB 到几 MB 之间，访问延迟为 10-20 个 CPU 周期。L3 缓存（也称为末级缓存，LLC）在多核处理器中通常是共享的，容量可达几十 MB，访问延迟为 30-60 个 CPU 周期[12]。主内存（DRAM）容量在 GB 级别，但访问延迟高达 100-300 个 CPU 周期。

这种层次化设计利用了程序访问的局部性原理：时间局部性（最近访问过的数据很可能在不久后再次被访问）和空间局部性（访问某一位置的数据后，很可能会访问其附近的数据）。通过在多级缓存中保存最频繁访问的数据，系统能够显著提高内存访问效率。

2.2 eBPF 技术基础

eBPF（Extended Berkeley Packet Filter）是 Linux 内核中的一项革命性技术，它允许在内核中安全地执行用户定义的程序，而无需修改内核源代码或加载内核模块。eBPF 起源于 BPF（Berkeley Packet Filter）[13]，后者是一种用于网络数据包过滤的技术[6]。1992 年，Steven McCanne 和 Van Jacobson 首次提出 BPF，它允许用户空间程序指定过滤规则，内核根据这些规则决定哪些网络数据包应该传递给用户空间。2014 年，Alexei Starovoitov 对 BPF 进行了彻底的重新设计，并将其扩展为 eBPF。这次升级大幅增强了 BPF 的功能：将指令集从原来的 2 个 32 位寄存器扩展到 10 个 64 位寄存器[4]，添加了复杂的数据结构支持（映射、哈希表等），并扩展了应用范围，从最初的网络数据包过滤扩展到跟踪、性能分析、安全等多个领域。

2.3 Linux perf 工具概述

Linux perf（也称为 perf_events）是 Linux 内核自带的性能分析工具，最初由 Ingo Molnar 开发，于 2009 年在 Linux 2.6.31 中引入[3]。perf 工具集提供了一系列命令行工具，用于收集、分析和可视化系统性能数据。

perf 的核心能力来自于内核的 perf_events 子系统，它提供了统一的接口来访问 CPU 的硬件性能计数器、软件计数器和跟踪点。perf 可以监控各种性能事件，包括：硬件事件：CPU 周期、指令执行、缓存命中/缺失、分支预测等；软件事件：上下文切换、次/主缺页、CPU 迁移等；跟踪点事件：系统调用、内核函数调用、文件 I/O 等。

2.4 本章小结

本章全面阐述了与内存性能监测相关的背景知识和技术基础。首先介绍了现代计算机系统的内存层次结构，从 CPU 寄存器到外部存储设备，详细分析了各级缓存的工作原理、性能特性以及关键指标。接着深入探讨了虚拟内存系统的实现，包括 TLB 机制和缺页异常处理过程，这些是理解内存访问性能的核心概念。

随后，本章系统性地介绍了 eBPF 技术，包括其发展历程、核心架构和主要组件。在性能监控应用方面，eBPF 凭借其低开销、高精度和可编程性优势，为新一代性能监测工具提供了技术支撑。

最后，本章概述了 Linux 平台上现有的性能监控工具，重点分析了 perf 工具的原理、架构和内存监控能力。

第三章 eMemGazer 访存性能监测工具系统设计与原理

本章将深入探讨 eMemGazer 系统的核心原理，从 eBPF 监控机制到与传统 perf 工具的对比分析，全面阐述该系统的技术基础和设计理念。

3.1 eBPF 监控核心原理

3.1.1 eBPF 在内核中的执行机制

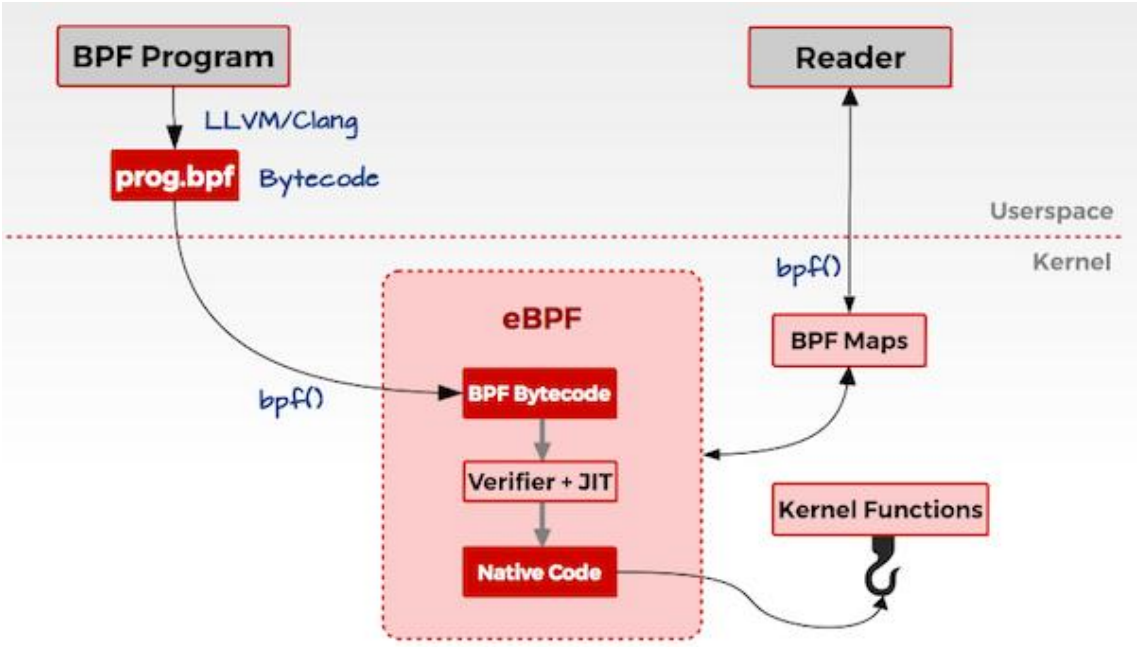


图 3.1　eBPF 执行机制原理图

eBPF 程序在 Linux 内核中的执行遵循一个严格定义的生命周期和安全执行模型，这是 eMemGazer 系统实现低开销、高精度内存性能监测的基础。当 eBPF 程序首次加载到内核时，它必须经过验证器的全面检查，确保程序不会导致内核崩溃或安全漏洞。这种验证过程包括检测无限循环、验证内存访问边界、分析程序路径复杂度以及确认程序会在有限时间内终止。

验证通过后，eBPF 字节码会通过即时编译器（JIT）转换为本地机器代码，显著提高执行效率[10]。现代 Linux 内核中 eBPF 程序的执行性能接近于原生内核代码，这使得它成为性能敏感场景下的理想技术选择。在 eMemGazer 中，我们充分利用这一特性，确保监测系统对被监测程序的性能干扰降至最低。

eBPF 程序的执行触发方式根据其附加点的不同而异。在 eMemGazer 中，我们主要使用两种类型的触发机制：基于性能事件的触发和基于跟踪点的触发。对于缓存访问、TLB 操作等硬件相关事件，程序在硬件性能计数器达到预设阈值时触

发执行；而对于缺页异常等内核事件，程序在相应的内核跟踪点被触发时执行。这种基于事件的执行模型保证了 eBPF 程序只在必要时运行，进一步降低了系统开销。

eBPF 程序执行的上下文包含丰富的信息，程序可以通过辅助函数和映射结构与内核交互，获取当前进程信息、访问内存地址、记录时间戳等。在 eMemGazer 中，我们精心设计了每个 eBPF 程序的执行逻辑，确保能够准确捕获所需的性能数据，同时避免不必要的计算或存储操作。

3.1.2 硬件性能计数器与 eBPF 的集成

现代处理器中的硬件性能监控单元（PMU）提供了大量可编程的性能计数器，这些计数器可以跟踪处理器微架构事件，如指令执行、分支预测、缓存操作和内存访问。Linux 内核通过 perf_events 子系统提供了统一的接口访问这些硬件计数器，而 eBPF 程序可以无缝集成这一机制，实现对硬件性能事件的精确监控。

在 eMemGazer 系统中，我们通过 perf_event_open 系统调用创建性能事件描述符，并将 eBPF 程序附加到这些事件上。具体实现中，我们利用 BPF_PROG_TYPE_PERF_EVENT 程序类型，使 eBPF 程序能够在性能事件触发时执行。系统监控的核心硬件事件包括：L1 数据缓存加载和未命中事件、末级缓存（LLC）加载和未命中事件、数据 TLB 加载和未命中事件、CPU 周期和指令执行事件。

3.1.3 内核跟踪点与缺页监控

除了硬件性能计数器，eMemGazer 还利用 Linux 内核提供的跟踪点（Tracepoints）机制监控缺页异常等软件事件[8]。跟踪点是内核中预定义的静态标记点，它们在关键代码路径上提供了稳定的观测界面，不受内核版本变化的影响。

在 eMemGazer 的设计中，我们特别关注用户空间缺页异常的监控。系统通过附加到 exceptions/page_fault_user 跟踪点，捕获每次用户空间内存访问导致的缺页异常。与硬件事件不同，跟踪点事件提供了更丰富的上下文信息，包括触发缺页的内存地址、访问类型（读或写）以及错误码等。这些信息使 eMemGazer 能够区分不同类型的缺页异常，例如首次访问导致的强制性缺页与页面交换导致的主缺页。

我们还通过综合分析缺页异常与内存访问模式的关联，为用户提供更有价值的性能洞察。例如，通过跟踪缺页异常的时间分布和空间分布，eMemGazer 可以识别可能导致性能问题的大规模随机内存访问模式，并在可视化界面中直观地呈现这些模式。

在实现上，eMemGazer 使用 BPF_PROG_TYPE_TRACEPOINT 程序类型，确保跟踪点事件触发时 eBPF 程序能够高效执行[9]。为了最小化监控开销，程序只记录必要的计数和关键指标，而不是收集每个缺页事件的详细信息，这种设计选择平衡了数据精度和系统性能。

3.1.4 eBPF 映射与数据传输机制

eBPF 映射（Maps）是 eBPF 程序存储状态和与用户空间通信的关键机制。在 eMemGazer 系统中，我们精心设计了映射结构，优化了内核态和用户态之间的数据传输，确保高效、可靠的性能数据收集。

系统主要使用两种类型的 eBPF 映射：

哈希映射（BPF_MAP_TYPE_HASH）：用于存储每个进程的缓存指标数据。哈希映射以进程 ID（PID）为键，以包含多个性能计数器的结构体为值。这种设计使系统能够同时监控多个进程，并为每个进程维护独立的性能统计。哈希映射的使用还简化了进程退出时的资源管理，内核会自动清理不再使用的映射条目，避免内存泄漏。

任务关联映射（BPF_MAP_TYPE_HASH 用于存储 PID 与 task_struct 的关联）：这个辅助映射解决了从内核获取进程名称的挑战。由于在 eBPF 程序上下文中可能无法直接访问完整的进程信息，我们通过这个映射存储进程的 task_struct 指针，使用户空间组件能够解析并显示有意义的进程名称，而不仅仅是数字 ID。

为了确保数据传输的效率，eMemGazer 采用了原子操作更新性能计数器。例如，使用 sync_fetch_and_add 函数原子地递增事件计数，这避免了在多处理器环境中可能出现的数据竞争问题，同时最小化了同步开销。

用户空间程序通过 bpf 系统调用定期从映射中读取性能数据，计算派生指标（如缓存未命中率和 IPC），并更新显示。我们在数据收集频率的设计上进行了权衡，选择了合适的间隔（默认为 1 秒），既保证了数据的时效性，又避免了过于频繁的读取操作导致的额外开销。

此外，eMemGazer 还实现了智能的数据过滤机制，只传输和处理活跃进程的性能数据，忽略那些性能计数器值无变化的进程。这大大减少了在多进程系统中的数据处理负担，使系统能够在资源受限的环境中高效运行。

3.2 eBPF 程序与 perf-event 的交互

Linux 内核的 perf-event 子系统是硬件性能监控的核心基础设施，它提供了统一的接口访问各种性能事件，包括硬件性能计数器、软件事件和跟踪点[14]。

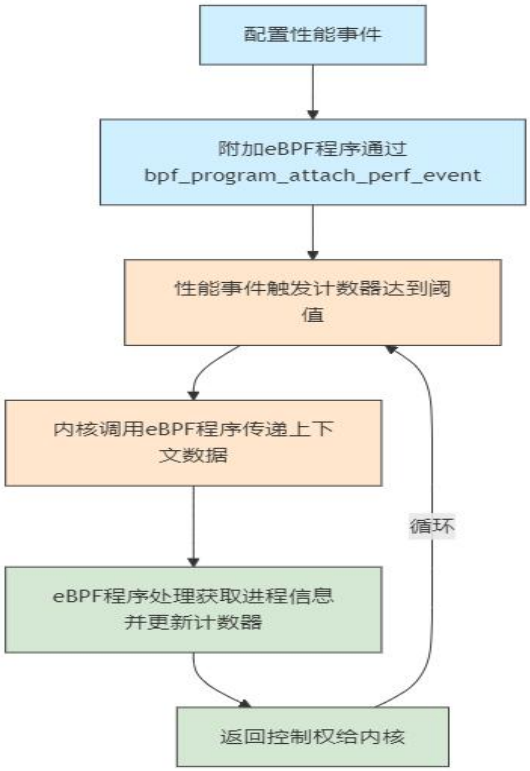


图 3.2　eBPF 程序与 perf-event 子系统交互图

eMemGaze 实现了 eBPF 程序与 perf-event 子系统的无缝集成。

eMemGazer 首先配置性能事件，设置适当的事件类型和参数。例如，对于 L1 数据缓存加载未命中事件，它使用 PERF_TYPE_HW_CACHE 事件类型，并指定适当的缓存级别（L1D）、操作类型（读取）和结果类型（未命中）。

一旦事件配置完成，系统通过 bpf_programattach_perf_event 函数将 eBPF 程序附加到性能事件上。这一步骤建立了事件与处理程序之间的连接，使程序能够在事件触发时执行。

当性能事件触发时，eBPF 程序会收到包含 struct bpf_perf_event_data 类型参数的调用。程序在此上下文中运行，获取当前进程信息，更新相应的性能计数器，然后返回控制权给内核。这种轻量级处理方式确保了监控过程的低开销。

3.3 与传统 perf 工具的对比分析

eMemGazer 与传统的 Linux perf 工具在架构设计和工作原理上有本质区别，这些差异直接影响了两者的性能特性和应用场景。

在架构方面，perf 采用经典的事件采样模型，通过配置的采样周期或频率，定期中断 CPU 并记录当前执行状态和性能计数器值。采样数据首先缓存在内核的环形缓冲区中，然后通过内存映射传输给用户空间进行处理和分析。

相比之下，eMemGazer 基于 eBPF 的事件触发模型，直接在内核中执行精简的数据收集逻辑。当性能事件发生时，eBPF 程序高效地更新内核中的计数器[11]，只在用户指定的报告间隔将聚合数据传输到用户空间。这种设计最小化了上下文切换和数据复制开销，显著降低了监控系统对被监测程序的干扰。

工作原理上，perf 主要采用统计采样方法，适合查找热点代码和性能瓶颈位置。它的优势在于提供全系统的性能概览，能够关联源代码层面的性能问题。但是，这种采样方法存在固有的精度限制，特别是对于短暂但频繁的事件难以准确捕获。

eMemGazer 则采用精确计数方法，专注于内存访问性能指标的准确统计。它不试图定位具体的热点代码行，而是提供进程级别的内存性能特征分析。这种方法在监控特定进程的内存访问行为时更为精确，能够捕获传统采样方法可能错过的细微性能模式。

两者在内存占用和 CPU 开销上也有明显差异。perf 在采样模式下会持续积两者在内存占用和 CPU 开销上也有明显差异。perf 在采样模式下会持续积累大量采样数据，这些数据需要足够的内存缓冲区存储，且随着采样频率的增加，内存和 CPU 开销也会显著增加。eMemGazer 通过在内核中进行数据聚合和过滤，大幅减少了传输到用户空间的数据量，即使在长时间监控的情况下也能保持较低的资源消耗。

值得注意的是，perf 和 eMemGazer 在底层都依赖于 Linux 内核的 perf_event 子系统访问硬件性能计数器，但使用方式有本质区别。perf 主要通过周期性采样读取计数器值，而 eMemGazer 则利用 eBPF 程序在性能事件触发时直接处理，避免了周期性中断和上下文切换的开销。这种差异在高频事件监控场景下尤为明显，eMemGazer 能够以更低的系统干扰提供更精确的性能数据。

3.4 本章小结

本章深入探讨了 eMemGazer 系统的核心原理，从 eBPF 监控机制到与传统 perf 工具的对比分析，全面阐述了该系统的技术基础和设计理念。我们首先分析了 eBPF 在内核中的执行机制，包括验证、JIT 编译和事件触发模型，这些机制构成了系统低开销、高精度监控的基础。随后详细讨论了硬件性能计数器与 eBPF 的集成方式，展示了系统如何利用 perf_event 机制精确捕获各级缓存、TLB 和 CPU 性能事件。

eBPF 映射与数据传输机制章节揭示了系统高效收集和传输性能数据的设计，

这是确保监控低开销的关键环节。通过对 eBPF 程序生命周期管理的详细分析，我们了解了系统如何实现程序的加载、验证、附加和资源释放，保证监控过程的可靠性和安全性。CO-RE 技术的应用展示了系统的跨内核版本兼容能力，这对于实际部署至关重要。

perf-event 与 eBPF 集成原理部分从内核实现角度分析了两者的交互机制，包括 perf-event 子系统架构、性能计数数据采集和处理流程，以及在数据采集过程中遇到的挑战与解决方案。

第四章 eMemGazer 访存性能监测工具系统实现

本章将阐述 eMemGazer 系统的设计与实现，从整体架构到具体模块，全面展示系统的技术路线和工程实践。系统采用分层架构，将内核态监测、用户态处理和可视化展示清晰分离，既保证了功能间的有效协作，也实现了各模块的独立演化能力。

4.1 系统架构设计

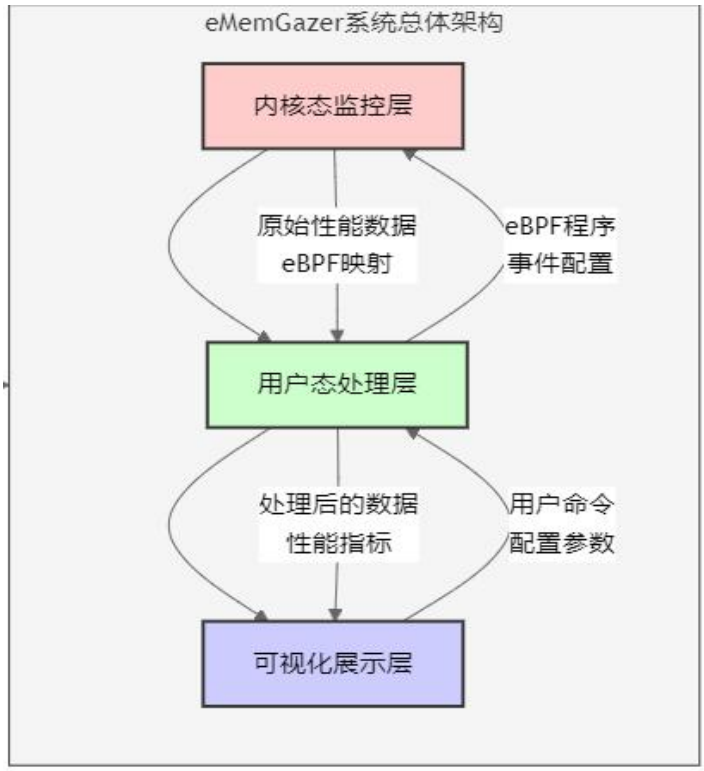


图 4.1　eMemGazer 系统总体架构图

eMemGazer 系统采用了分层模块化的架构设计。系统整体分为三个主要层次：内核态监控层、用户态处理层和可视化展示层，每个层次负责特定的功能并通过标准接口与其他层次交互。

内核态监控层是系统的基础，由一系列 eBPF 程序组成，负责直接与 Linux 内核交互，捕获并初步处理内存性能事件。这些 eBPF 程序战略性地附加在内核的关键位置，包括硬件性能事件触发点、内存管理跟踪点以及系统调用入口点。内核态组件的核心职责是高效地收集原始性能数据，执行必要的聚合和过滤，并通过 eBPF 映射机制将数据传递给用户态组件。

用户态处理层是系统的中枢，负责 eBPF 程序的生命周期管理、性能数据的收

集和处理以及与用户的交互。该层包含多个功能模块：程序加载器负责编译、验证和加载 eBPF 程序到内核；数据收集器定期从 eBPF 映射读取性能数据；数据处理器计算各种派生指标并准备可视化所需的数据结构；控制接口处理用户命令和配置，调整系统行为以适应不同的监测需求。

可视化展示层是系统的前端，将复杂的性能数据转化为直观的图表和报告，帮助用户理解内存访问行为并识别性能瓶颈。该层采用了现代前端技术栈，实现了多维度的数据展示，包括实时监控视图、历史趋势分析、不同内存访问模式的对比分析等。系统支持多种展示形式，从命令行输出到交互式 Web 界面，满足用户的需求。

这三个层次通过精心设计的接口协同工作，形成完整的数据流：内存性能事件首先被内核态组件捕获，经过初步处理后传递给用户态组件，用户态组件进一步加工处理这些数据，最后由可视化组件以用户友好的方式呈现。这种架构确保了系统具有高度的模块化和可扩展性，各组件可以独立演化，同时保持整体系统的稳定性和一致性。

4.2 内核态监测模块实现

内核态监控模块是与 Linux 内核直接交互的组件集合，包含以下子模块：

性能事件监控器：附加到硬件性能计数器事件，捕获缓存访问、TLB 操作和 CPU 执行事件。该模块负责区分不同类型的性能事件，并将事件信息与触发进程关联。

缺页异常跟踪器：通过附加到内核跟踪点，监控用户空间缺页异常的发生。这个模块能够识别不同类型的缺页异常（如次缺页和主缺页），并记录相关统计信息。

数据汇集器：在内核中对收集的性能数据进行初步聚合和过滤，减少需要传输到用户空间的数据量。该模块使用原子操作更新共享计数器，确保在多核环境下的数据一致性。

映射管理器：维护 eBPF 映射结构，处理数据存储和访问控制。该模块实现了高效的键值查找和更新机制，优化了内核态和用户态之间的数据共享。

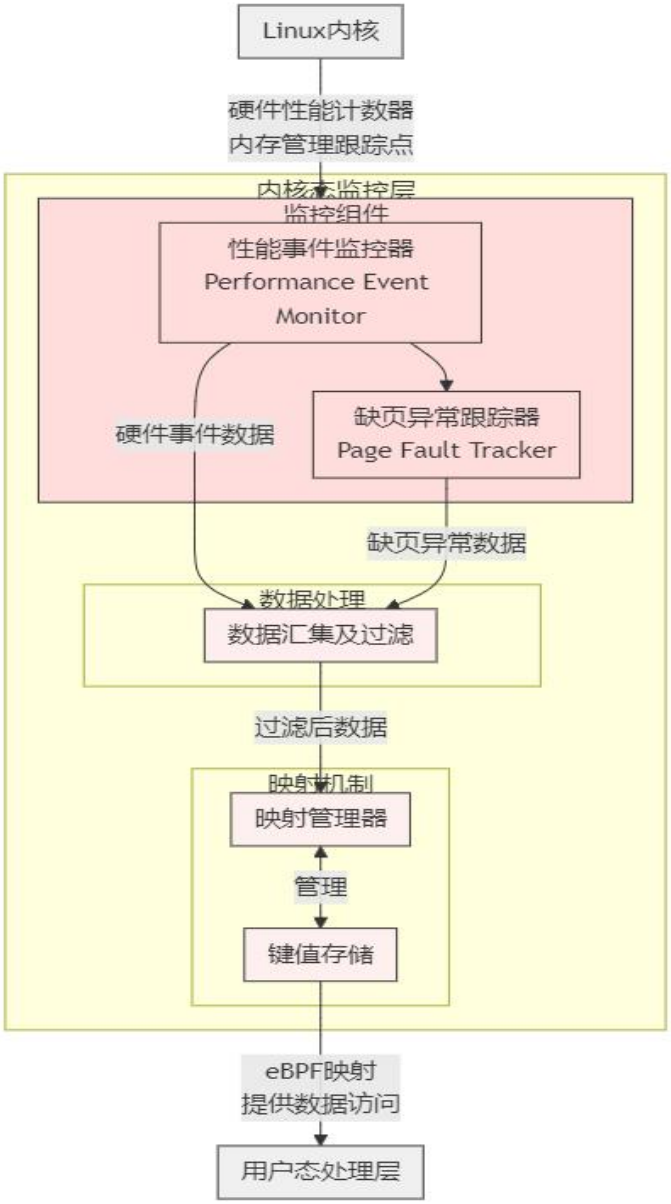


图 4.2　eMemGazer 内核态监控模块图

4.2.1 性能事件捕获机制

eMemGazer 的内核态监测组件实现了精确、高效的性能事件捕获机制，这是系统准确监测内存访问性能的基础。该机制基于 Linux 内核的 perf_event 子系统和 eBPF 技术，通过精心设计的程序结构和事件配置，实现了对各类硬件和软件性能事件的精确捕获。

系统采用事件类型分离的设计原则，更容易通过内核的 eBPF 验证器检查。主要的事件捕获程序包括 L1 数据缓存加载/未命中处理器、末级缓存(LLC)加载/未命中处理器、数据 TLB 加载/未命中处理器、CPU 周期和指令执行处理器以及缺页异

常处理器等。

对于硬件性能事件，系统通过 perf_event_open 系统调用创建性能事件描述符，并配置适当的事件类型和属性。例如，对于 L1 数据缓存未命中事件，系统使用 PERF_TYPE_HW_CACHE 类型，并设置特定的缓存类型(PERF_COUNT_HW_CACHE_L1D)、操作类型(PERF_COUNT_HW_CACHE_OP_READ)和结果类型(PERF_COUNT_HW_CACHE_RESULT_MISS)。这种精确配置允许系统准确捕获特定类型的缓存操作和结果。

事件描述符创建后，系统通过 bpf_programattach_perf_event 函数将 eBPF 程序附加到相应的事件上。当事件触发时（例如，当 L1 缓存未命中发生时），内核会执行附加的 eBPF 程序。程序首先通过 bpf_get_current_pid_tgid()获取当前进程 ID，确定哪个进程触发了该事件。然后，程序查找或创建该进程的性能指标记录，并原子地更新相应的计数器。

对于缺页异常等内核事件，系统使用跟踪点机制。eMemGazer 对于缺页异常等内核事件，系统使用跟踪点机制。eMemGazer 附加到 exceptions/page_fault_user 跟踪点，这是内核处理用户空间缺页异常的关键位置。当程序访问未加载到物理内存的虚拟内存地址时，处理器会触发缺页异常，内核进入相应的异常处理流程。在这个处理流程中，如果异常源于用户空间内存访问，则会触发 exceptions/page_fault_user 跟踪点。附加的 eBPF 程序会捕获这一事件，记录触发缺页的进程信息和计数。

性能事件捕获机制的一个关键优化是上下文信息的精确获取。eBPF 程序可以通过内核提供的辅助函数获取丰富的执行上下文，包括当前进程信息、CPU 状态和内存访问特性。例如，在缺页异常处理中，程序可以获取触发异常的内存地址和访问类型，这些信息对于理解程序的内存访问模式至关重要。

为了确保事件捕获的准确性，系统对硬件性能计数器进行了特殊的配置。默认情况下，性能计数器只计数基本事件，可能无法捕获某些细粒度事件。eMemGazer 通过设置适当的性能事件属性，如排除内核模式计数、启用精确事件采样等，确保获取到的是用户空间程序真实的内存访问特征，而不受内核活动的干扰。

系统还实现了智能的事件处理优先级管理。硬件性能事件通常数量有限，当需要监控的事件数量超过硬件支持的数量时，内核会使用时间复用（time multiplexing）机制。eMemGazer 识别这种情况，并通过调整事件属性和处理逻辑，最小化复用对计数准确性的影响。系统还会记录实际运行时间与启用时间的比率，用于后续数据处理时对计数值进行校正。

4.2.2 eBPF 映射实现与优化

eBPF 映射是 eMemGazer 内核态与用户态组件之间共享数据的关键机制，系统对其进行了精心设计和优化，确保高效、可靠的性能数据传输。eMemGazer 主要使用两种类型的映射：性能指标映射和进程信息映射，它们共同构成了系统的数据共享基础设施。

性能指标映射（process_metrics）是一个哈希表类型的 BPF 映射，用于存储各进程的内存性能指标。该映射以进程 ID（PID）作为键，以包含多个性能计数器的结构体（struct cache_metrics）作为值。这种设计允许系统高效地按进程分类存储和检索性能数据，满足进程级精细监控的需求。性能指标结构包含了 L1 数据缓存加载/未命中次数、末级缓存加载/未命中次数、TLB 操作统计、缺页异常计数以及 CPU 周期和指令执行计数等关键性能指标。

进程信息映射（task_pid）是系统的辅助映射，设计用于解决内核和用户空间之间的进程信息共享问题。该映射同样使用哈希表类型，以 PID 为键，以指向 task_struct 的指针（用户空间视为 u64 类型）为值。这种设计允许用户空间程序获取进程的完整信息，如进程名称、父进程 ID 等，这些信息对于生成可读性良好的监控报告至关重要。

系统对 eBPF 映射实现了多项关键优化：

首先是映射大小的优化。eBPF 映射在创建时需要指定最大条目数，这个数值需要权衡内存使用和可监控进程数量。过大的映射会消耗过多内核内存，而过小的映射则可能无法容纳所有活跃进程的性能数据。通过分析目标系统的典型工作负载特性，eMemGazer 将映射大小设置为 8192 条目，能够满足大多数使用场景的需求，同时避免过度消耗内存资源[15]。

其次是映射访问模式的优化。在 eBPF 程序中，映射查找和更新操作是最常见的操作，也是潜在的性能瓶颈。eMemGazer 采用了"查找或初始化"模式，即首先尝试查找现有条目，如果不存在则创建并初始化新条目。这种模式减少了冗余的映射操作，提高了程序执行效率。为了进一步优化性能，系统在 eBPF 程序中使用了局部变量缓存映射查找结果，避免在单次事件处理中重复查找相同的键。

第三是原子操作优化。由于多个 CPU 核心可能同时处理来自同一进程的性能事件，对共享计数器的更新需要保证原子性，避免数据竞争和不一致。eMemGazer 使用 sync_fetch_and_add 等原子指令更新性能计数器，确保计数的准确性，同时避免了锁操作带来的高开销。

第四是内存占用优化。为了减少每个映射条目的内存占用，系统对性能指标结构进行了紧凑设计，确保字段对齐且无冗余填充。这种优化虽然看似微小，但在大规模监控场景下，可以显著减少系统的内存消耗，提高资源使用效率。

最后是映射清理策略优化。在长时间运行的监控会话中，映射可能累积大量已退出进程的条目，浪费宝贵的映射空间。eMemGazer 实现了智能清理策略，用户空间组件定期检查映射中的进程是否仍然存在，并移除已退出进程的条目，释放映射空间以容纳新的活跃进程。这种主动清理确保了映射资源的高效利用，即使在系统进程频繁创建和退出的环境中也能稳定运行。

通过这些精心设计的映射结构和优化策略，eMemGazer 实现了高效、可靠的内核态和用户态数据共享，为系统的低开销特性和准确的性能监测能力奠定了坚实基础。

4.2.3 内核数据结构访问

eMemGazer 系统的一个核心技术挑战是如何在 eBPF 程序中安全、高效地访问 Linux 内核的关键数据结构。

在 Linux 内核中，进程管理是通过 task_struct 结构体实现的，这个结构体包含了进程的所有关键信息。eMemGazer 需要访问这个结构体以获取进程的标识符、名称、内存使用状态等信息。系统通过 bpf_get_current_task()辅助函数获取当前执行上下文的 task_struct 指针，这是一个由内核提供的安全方法，确保 eBPF 程序只能访问当前执行环境的任务信息，而不能任意访问系统中的其他进程。

获取 task_struct 指针后，系统需要从中提取所需的信息。然而，在 eBPF 程序中直接访问复杂结构体的成员存在挑战，因为内核的 eBPF 验证器需要确保所有内存访问都在安全边界内。eMemGazer 采用了多种技术解决这一问题：

首先，系统使用 CO-RE（Compile Once – Run Everywhere）技术处理内核数据结构在不同版本间的变化。通过 BPF_CORE_READ 宏和相关机制，程序能够在加载时动态调整对结构成员的访问，适应不同内核版本中字段位置可能发生的变化。这种方法避免了为每个目标内核版本单独编译程序的需要，显著提高了系统的可移植性。

其次，对于需要频繁访问的关键信息，如进程 ID 和命名空间信息，系统实现了辅助映射存储机制。eBPF 程序将这些关键数据保存在专用映射中，用户空间程序可以直接从映射读取，而不需要在每次数据收集时重新从内核结构中提取。这种设计不仅提高了数据访问效率，也简化了程序逻辑，使 eBPF 程序更容易通过验证器检查。

内核数据结构访问的一个特殊挑战是处理内核中的链表和树形结构。这些结构通常用于管理复杂的关系，如进程的内存映射区域列表。由于 eBPF 程序中的循环受到严格限制，传统的遍历方法无法直接应用。eMemGazer 通过精心设计的有界循环和循环展开技术，实现了对这些复杂结构的有限但有效的访问，在确保程

序通过验证器的同时，获取了监控所需的关键信息。

通过这些先进的内核数据结构访问技术，eMemGazer 能够在不影响系统安全性的前提下，获取丰富的性能上下文信息，为准确分析内存访问行为提供了坚实的数据基础。

4.2.4 安全性与性能平衡

在设计和实现 eMemGazer 系统时，安全性与性能之间的平衡是一个核心考量。eBPF 作为一种运行在内核中的技术，其安全性直接关系到整个系统的稳定性，而性能监测工具的低开销要求又使得性能优化至关重要。eMemGazer 通过精心设计的安全机制和性能优化策略，在两者之间取得了出色的平衡。

在安全性方面，eMemGazer 严格遵循 eBPF 的安全模型，这一模型基于静态验证和受限执行环境。内核的 eBPF 验证器会在程序加载时进行全面的安全检查，包括控制流分析、内存访问验证和资源使用限制[10]。为了确保 eMemGazer 的 eBPF 程序能够顺利通过这些严格的检查，系统采用了多项设计策略：

首先，程序结构保持简单清晰，避免复杂的控制流和深层嵌套，使验证器能够有效分析所有可能的执行路径。每个 eBPF 程序专注于单一职责，如处理特定类型的性能事件，这不仅简化了程序逻辑，也降低了潜在的安全风险。

其次，内存访问模式经过精心设计，确保所有指针操作都在安全边界内。系统对每次内存访问都进行边界检查，使用辅助函数和内核提供的安全 API 访问内核数据结构，避免直接操作原始内存地址，这降低了内存损坏和信息泄露的风险。

第三，资源使用受到严格控制。eBPF 程序的栈大小有限（通常为 512 字节），系统确保所有局部变量和函数调用不会超出这一限制。同样，系统也控制了映射的大小和数量，避免过度消耗内核内存资源。

最后，eMemGazer 遵循最小权限原则，只请求完成任务所需的最小权限集合。例如，系统只在必要时使用特权辅助函数，并且所有对内核数据的访问都遵循既定的安全接口，不尝试绕过内核的访问控制机制。

在性能优化方面，eMemGazer 采用了多层次的策略，在保证安全的前提下最小化监控开销：

第一层是事件过滤优化。不是对每次内存访问都触发 eBPF 程序执行，系统通过配置性能事件的采样参数，控制程序触发频率，平衡数据精度和执行开销。对于不同类型的性能事件，系统根据其特性和重要性设置不同的采样策略。

第二层是执行路径优化。eBPF 程序的执行路径经过精心设计，最小化指令数量和分支操作。系统利用 BPF 编译器和 JIT 机制的特性，编写对 CPU 架构友好的代码，减少不必要的计算和内存访问。例如，使用寄存器变量存储中间结果，避

免重复的映射查找操作。

第三层是数据处理优化。系统在内核中只进行必要的计数和基本聚合，复杂的数据处理和分析推迟到用户空间执行。这种设计减少了 eBPF 程序的复杂度和执行时间，同时也满足了安全要求，因为验证器更容易分析和验证简单的程序逻辑。

第四层是上下文切换优化。通过合理设计事件处理逻辑和映射访问模式，系统最小化了内核和用户空间之间的上下文切换次数。数据收集采用批处理模式，每次用户空间读取会获取多个进程的数据，减少系统调用开销。

在实际部署中，eMemGazer 允许用户根据具体需求调整安全和性能参数，如监控的进程范围、数据收集频率和详细程度等。这种灵活性使系统能够适应从高安全要求的生产环境到高性能需求的开发测试环境等各种场景。

4.3 用户态数据处理模块实现

用户态处理模块负责 eBPF 程序的管理和数据处理，包括：

eBPF 加载器：处理 eBPF 程序的编译、验证和加载。该模块实现了 CO-RE 兼容性逻辑，确保程序能在不同内核版本上正确运行。

性能事件配置器：负责创建和配置性能事件，处理事件多路复用，并管理事件与 eBPF 程序的关联。

数据收集与处理器：定期从 eBPF 映射中读取性能数据，计算派生指标，并准备用于展示的数据结构。该模块实现了数据缓存和增量处理逻辑，优化了数据处理效率。

命令行界面：提供用户交互接口，处理配置参数，显示监控结果，支持各种命令行选项以满足不同监控需求。

4.3.1 eBPF 程序管理

eMemGazer 的用户态组件负责管理 eBPF 程序的整个生命周期，从加载到执行再到卸载，确保内核态监测组件的正确运行。这一管理机制是系统稳定性和可靠性的关键保障，它通过精心设计的程序管理框架实现了对复杂 eBPF 程序集合的有效控制。

程序管理的核心是基于 libbpf 库构建的骨架（Skeleton）管理机制。libbpf 是一个高级用户空间库，提供了加载和管理 eBPF 程序的 API，隐藏了底层 BPF 系统调用的复杂性。eMemGazer 利用 libbpf 的骨架机制，通过自动生成的骨架代码（通常由 BPF 编译系统自动生成）创建程序和映射对象的类型安全表示。

程序加载过程采用三阶段模型：打开、加载和附加。首先，系统调用 cache_monitor_bpfopen()函数打开 BPF 骨架，初始化内部数据结构并准备程序对象。

在这个阶段，系统可以根据运行时配置调整程序参数，如映射大小或全局变量值。接着，调用 cache_monitor_bpf_load()将 eBPF 对象加载到内核，触发验证过程。最后，系统将程序附加到相应的事件源，使其能够响应监控事件。

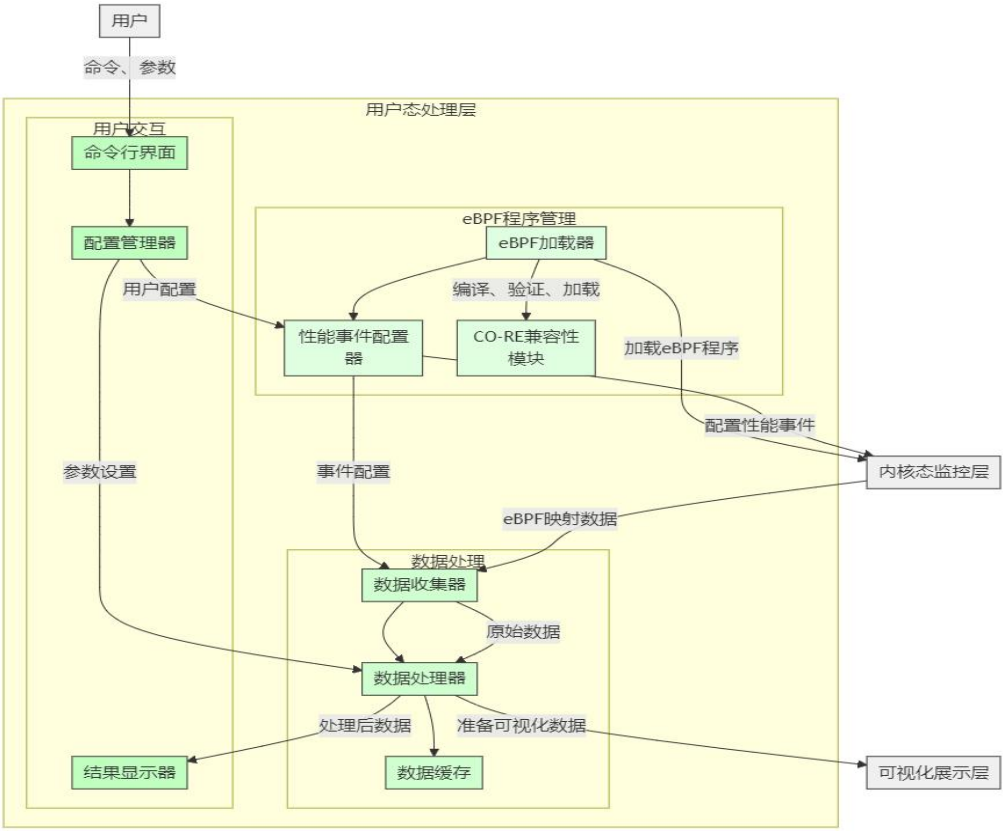


图 4. 2 eMemGazer 用户态处理模块图

为了处理不同内核版本和硬件平台的兼容性问题，eMemGazer 实现了智能的探测和适配机制。系统首先检测目标环境支持的硬件性能事件和内核特性，然后根据检测结果调整程序加载和配置策略。例如，如果特定的缓存事件在目标平台上不可用，系统会跳过相应的程序附加步骤，而不是导致整个系统加载失败。这种灵活性使 eMemGazer 能够在各种环境中提供最佳可能的功能集。

错误处理是程序管理中的关键环节。eBPF 程序加载和附加过程中可能遇到各种错误，如验证失败、资源不足或权限问题。eMemGazer 实现了全面的错误检测和处理机制，捕获各种错误情况并提供有意义的错误信息。系统对关键错误（如全部程序加载失败）和非关键错误（如部分事件不可用）区别对待，确保在各种情况下都能提供最佳可能的功能。

资源管理和清理是程序管理的另一个重要方面。eBPF 程序在内核中运行，消耗内核资源，如果不正确释放，可能导致资源泄漏。eMemGazer 实现了强健的资源管理策略，使用对象生命周期管理模式，确保所有资源在不再需要时得到适当

释放。系统注册信号处理函数，捕获 SIGINT 和 SIGTERM 信号，以便在程序被中断时执行清理操作。清理过程包括销毁所有 eBPF 链接、关闭性能事件文件描述符和释放 BPF 骨架资源，确保系统退出时不留下任何活动的内核组件。

程序管理机制还包括状态监控和健康检查功能。系统定期检查已加载程序的状态，确保它们仍然正常运行。如果检测到程序失效或异常行为，系统会尝试重新附加或重新加载程序，确保监控功能的连续性。这种自愈能力使 eMemGazer 能够在长时间运行时保持稳定性，即使在面对系统负载变化和资源竞争的情况下也是如此。

通过这种全面而灵活的程序管理机制，eMemGazer 实现了对 eBPF 程序的高效、可靠控制，为系统的低开销特性和强大功能提供了坚实基础。这种机制不仅简化了系统的使用和维护，也为未来扩展新的监控能力提供了灵活的框架。

4.3.2 性能数据收集与处理

eMemGazer 的用户态组件负责从内核收集原始性能数据，并通过多阶段处理转换为有意义的性能指标。这一数据收集与处理机制是系统分析能力的核心，通过精心设计的处理流水线，实现了高效、准确的内存性能分析。

数据收集过程采用定时器驱动的模型。系统创建一个主循环，在用户指定的间隔（默认为 1 秒）周期性地从 eBPF 映射读取性能数据。这种基于间隔的设计平衡了数据时效性和系统开销，确保能够捕获性能变化趋势，同时避免过度频繁的数据收集导致的高开销。系统还支持用户通过命令行参数调整收集间隔，以适应不同的监控需求，从高频实时监控到低频长期趋势分析。

数据收集机制实现了智能的过滤策略，只处理活跃进程的数据。系统使用多级过滤机制：首先通过查询/proc 文件系统检查进程是否仍然存在，过滤掉已退出的进程；然后通过比较当前和上一次读取的性能计数器值，过滤掉计数器无变化的进程；最后，如果用户指定了目标 PID，系统只处理该进程的数据，忽略其他进程。这种多级过滤显著减少了需要处理的数据量，提高了系统效率。

数据处理过程包含多个关键步骤。首先，系统从 eBPF 映射读取原始性能计数器值，这些值包括各级缓存的访问和未命中计数、TLB 操作统计、缺页异常计数以及 CPU 周期和指令执行计数等。接着，系统计算性能计数器的增量，即相比上次读取的变化值。这些增量反映了当前监控间隔内的性能活动，是计算瞬时性能指标的基础。

关键派生指标的计算是数据处理的核心步骤。系统计算多种性能比率：

L1 数据缓存未命中率 = (L1 数据缓存未命中次数 / L1 数据缓存访问次数) × 100%

末级缓存(LLC)未命中率 =(LLC 未命中次数 / LLC 访问次数) × 100%

数据 TLB 未命中率 =(数据 TLB 未命中次数 / 数据 TLB 访问次数) × 100%

每周期指令数(IPC) = 指令执行次数 / CPU 周期数

计算这些指标时，系统特别注意处理边缘情况，如分母为零或极小值、计数器溢出等，确保计算结果的准确性和稳定性。

系统还处理硬件计数器的特殊情况，如多路复用和采样偏差。在多核系统上，性能计数器可能不能同时监控所有事件，导致某些事件只在部分时间内被监控。eMemGazer 通过检查计数器的 time_enabled 和 time_running 比率，对原始计数进行缩放校正，减少这种多路复用导致的计数偏差：

校正后的计数 = 原始计数 × (time_enabled / time_running)

数据处理的最后阶段是格式化和组织结果。系统将处理后的数据组织成结构化格式，准备用于显示或进一步分析。根据用户配置，系统可以生成不同格式的输出，包括人类可读的表格格式、机器可解析的 JSON 或 CSV 格式，以及用于可视化的数据结构。

此外，系统实现了数据聚合功能，支持按不同维度聚合性能数据。在进程级监控中，系统聚合同一进程的所有线程的性能数据；在系统级监控中，系统可以聚合所有进程或特定进程组的数据，提供系统整体性能视图。这种多级聚合能力使 eMemGazer 能够从不同粒度分析内存性能，满足从微观到宏观的各种分析需求。

通过这种全面而精细的数据收集和处理机制，eMemGazer 将原始的性能计数器数据转化为有意义的性能指标，为用户提供深入的内存性能洞察，帮助识别性能瓶颈和优化机会。

4.4 数据可视化模块实现

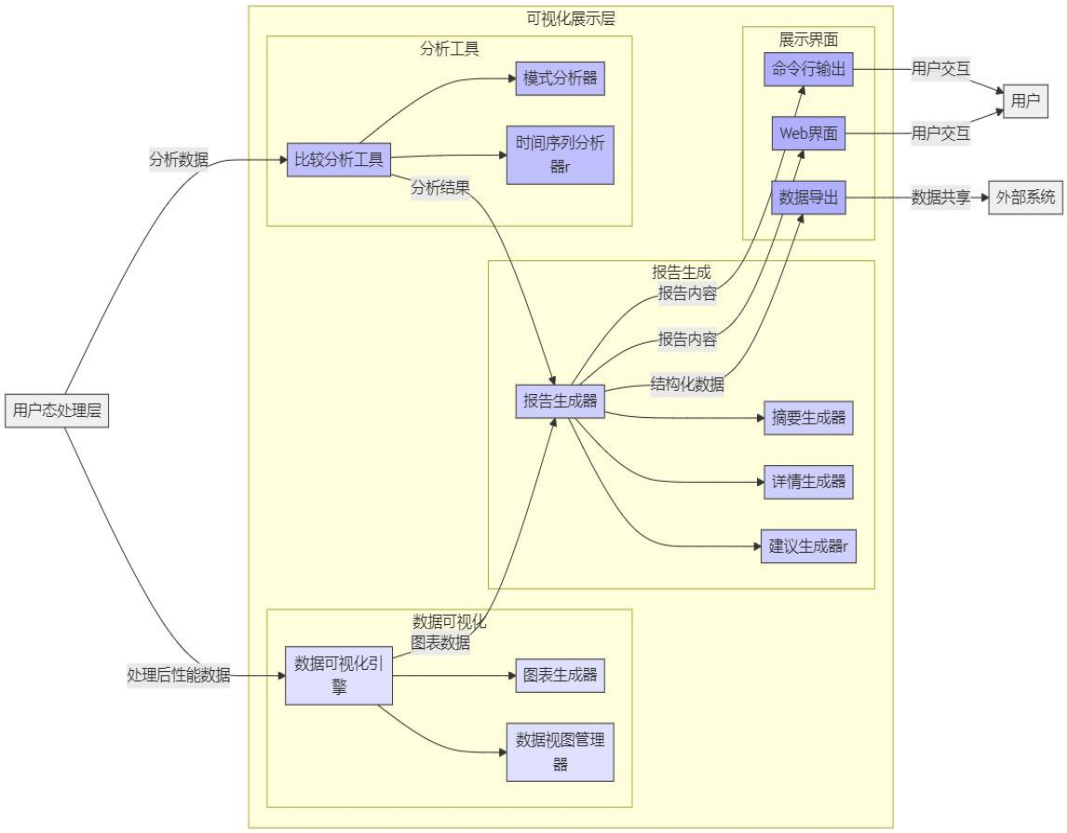


图 4.3　eMemGazer 可视化展示模块图

可视化展示模块将性能数据转化为直观的视觉表现，包括：

数据可视化引擎：将处理后的性能数据转换为各种图表和视图，如柱状图、折线图和雷达图等。该模块支持交互式操作，如数据过滤、聚合和细节查看。

报告生成器：创建综合性能报告，包括摘要统计、详细分析和优化建议。支持多种输出格式，如 HTML、JSON 和 CSV。

比较分析工具：实现不同内存访问模式或不同监控工具结果的对比分析，帮助用户理解不同方法的性能特征和差异。

测试支持模块提供自动化测试：

内存访问模式生成器：实现多种典型的内存访问模式，如顺序访问、随机访问和步长访问，用于验证和评估监控系统。

可视化组件的核心是比较分析框架，它支持多种比较维度，包括不同内存访问模式之间的比较、eMemGazer 与 perf 等其他工具结果的对比、不同硬件配置下的性能对比等。这种比较能力使用户能够理解不同条件下内存访问性能的变化，为优化决策提供依据。

在内存访问模式比较方面，系统展示了顺序访问、随机访问和步长访问这三种典型模式下的性能特征。每种模式产生不同的缓存行为和内存访问模式：顺序

访问通常具有高缓存命中率，体现良好的空间局部性；随机访问则表现出高缓存未命中率，几乎没有局部性；步长访问在两者之间，其性能特征取决于步长与缓存行大小的关系。系统通过柱状图和雷达图等形式，直观展示这些模式在各项性能指标上的差异，帮助用户理解程序访问模式对性能的影响。

监控工具比较是另一个重要功能，系统自动执行 eMemGazer 与 Linux perf 工具的对比测试，将两者在相同工作负载下的监测结果进行对比分析。这种比较不仅验证了 eMemGazer 数据的准确性，也展示了两种工具在不同方面的优势。比较结果通过并排柱状图和差异百分比表格呈现，使用户能够直观了解两种工具的测量差异。

可视化组件支持多种图表类型，每种类型适合展示特定类型的性能数据：

柱状图用于比较不同条件下的静态性能指标，不同访问模式的缓存未命中率。

折线图展示性能指标随时间变化趋势，适合监控长时间运行的程序性能演变。

雷达图（也称为蜘蛛图）同时展示多个性能指标，提供性能特征的整体视图。

堆叠图展示组成关系，如不同类型的缓存未命中在总未命中中的占比。

所有图表都实现了交互式特性，用户可以通过鼠标悬停查看详细数据值，通过点击显示或隐藏特定数据系列，通过缩放和平移探索大量数据。这种交互性增强了可视化的信息密度和探索能力。

报告生成功能将性能分析结果组织为结构化报告，适合存档、共享和后续分析。系统支持多种报告格式，包括 HTML（适合 Web 浏览和交互）、CSV（适合数据导入到其他分析工具）和纯文本（适合命令行环境和邮件分享）。HTML 报告特别丰富，包含交互式图表、详细的性能数据表格和分析结论。

在可视化实现技术上，系统采用了 Chart.js 库创建交互式图表，这一库提供了丰富的图表类型和灵活的定制选项，同时保持了轻量级特性，适合集成到性能监控工具中。系统生成的 HTML 报告使用响应式设计，能够自适应不同显示设备，从桌面监视器到移动设备都能提供良好的查看体验。为了确保可视化结果的一致性和专业性，系统定义了统一的颜色方案和布局模板，所有图表和报告都遵循这些设计规范。

4.5 本章小结

本章详细阐述了 eMemGazer 系统的设计与实现，内核态监测组件基于 eBPF 技术，实现了对硬件性能事件和内核跟踪点的精确捕获，通过优化的映射结构和访问机制，在确保安全性的同时实现了高效的性能数据收集。

用户态处理组件负责 eBPF 程序的生命周期管理和性能数据的收集处理，通过

精心设计的命令行界面，提供了灵活、强大的监控配置和控制能力。系统实现了完善的数据可视化和报告生成功能，将复杂的性能数据转化为直观的图表和分析结果，帮助用户获取有价值的性能洞察。比较测试框架则提供了自动化测试和评估能力，通过模拟不同的内存访问模式，系统地比较不同监控工具的性能测量结果，验证了 eMemGazer 的准确性并展示了不同内存访问策略的性能特征。

整个系统实现过程中特别注重安全性与性能的平衡，通过多层次的优化策略，确保监控工具自身的开销最小化，不会显著干扰被监测程序的正常执行。系统还实现了全面的错误处理和资源管理机制，确保在各种环境和使用场景下的稳定性和可靠性。

第五章 系统测试与性能评估

5.1 测试环境与配置

5.1.1 硬件与操作系统环境

操作系统环境是 Linux 发行版 Ubuntu 22.04 LTS，搭载内核版本 6.15。这个内核版本包含完整的 eBPF 功能支持，包括 CO-RE（Compile Once – Run Everywhere）兼容性和高级 BPF 特性。系统配置了标准的性能调度器和默认的内存管理策略，代表了典型的服务器工作环境。为了确保测试的一致性，禁用了自动频率调节（禁用 intel_pstate 驱动的 turbo 模式）和透明大页（Transparent Huge Pages），这些功能可能导致性能波动，影响测试结果的可重复性。

为了评估 eMemGazer 在不同平台上的兼容性和性能特征，系统还在以下辅助测试环境中进行了验证：

每个测试环境都进行了标准化配置，包括禁用不必要的后台服务、统一的系统负载水平和一致的编译器优化设置（使用 GCC 11.2.0，优化级别为-O2）。所有测试在受控的温度环境中进行，确保热调节不会影响处理器性能和测试结果。

5.1.2 测试工具与方法论

测试工具集包括以下核心组件：

测试驱动程序：自主开发的 memory_test 程序是主要的工作负载生成工具，它实现了多种标准内存访问模式，包括顺序访问、随机访问和步长访问。该程序使用 64MB 大小的内存数组（确保远大于任何级别的处理器缓存），通过可配置的访问模式和重复次数，产生可控、可重复的内存访问负载。程序设计考虑了编译器优化的影响，确保生成的访问模式不会被现代编译器优化掉，真实反映预期的内存行为。

参考监控工具：Linux perf 工具作为性能监控的行业标准，用作 eMemGazer 结果的参考基准。测试使用相同的 perf_event 配置（如 L1-dcache-loads、L1-dcache-load-misses 等）监控同一工作负载，确保比较的公平性和有效性。perf 工具以统计模式运行，收集完整的事件计数而非采样数据，与 eMemGazer 的数据收集方式保持一致。

自动化测试框架：compare_mem_tests.sh 脚本实现了完整的测试自动化，包括环境准备、测试执行、数据收集和结果分析。该框架支持批量测试多种内存访问模式，自动对比 eMemGazer 和 perf 的结果，并生成标准化的报告和可视化输出。

测试方法论遵循以下核心原则：

控制变量法：在比较不同内存访问模式或不同监控工具时，严格控制所有其他变量保持不变。例如，在比较 eMemGazer 和 perf 时，使用完全相同的工作负载、运行时间和性能事件配置，确保差异仅来自监控工具本身。

重复测试与统计分析：每项测试重复执行多次（通常为 5-10 次），收集统计有效的数据集。测试结果报告包括平均值、标准差和变异系数，反映结果的中心趋势和稳定性。异常值通过标准的统计方法识别和处理，确保结果不受偶发性能波动的影响。

多维度性能指标：评估使用多种性能指标，不仅包括直接的缓存性能指标（如缓存未命中率），还包括衍生指标（如 IPC 值）和系统级指标（如 CPU 利用率）。这种多维度分析提供了工作负载性能特征的全面视图，避免了单一指标可能带来的片面评估。对于每种内存访问模式，系统收集并分析 L1 缓存、LLC 缓存、TLB 性能、缺页率以及指令执行效率等关键指标，形成完整的性能画像。

5.1.3 测试场景设计

系统测试了三种典型的内存访问模式：顺序访问、随机访问和步长访问。这些模式代表了内存访问局部性的不同极端情况，是评估缓存系统和内存子系统性能的标准测试模型。

顺序访问测试模拟连续的内存访问模式，是最优的缓存使用场景。测试程序按顺序遍历大型数组，每个元素访问一次，然后重复多次迭代。这种模式充分利用了空间局部性和硬件预取机制，通常表现为高缓存命中率和高指令执行效率。顺序访问代表了数据流处理、连续扫描和顺序文件访问等应用场景。

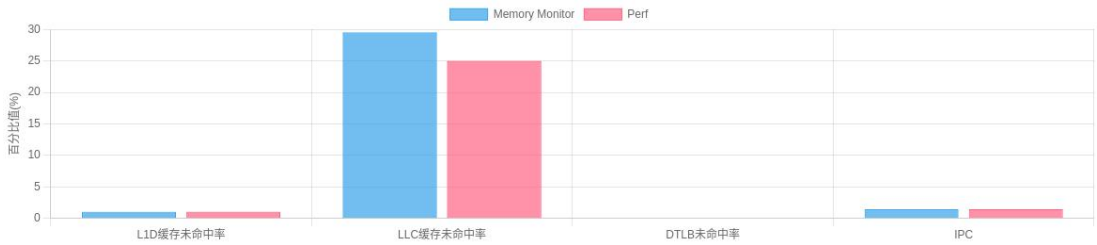
随机访问测试使用伪随机序列访问内存位置，是最差的缓存使用场景。测试程序使用高质量的随机数生成器决定访问的数组索引，确保访问分布均匀且不存在可被硬件预测的模式。这种访问模式几乎没有空间或时间局部性，通常导致高缓存未命中率和低指令执行效率。随机访问代表了哈希表查找、稀疏矩阵运算和随机数据库访问等应用场景。

步长访问测试使用固定步长（非连续）访问内存，是介于顺序和随机之间的中间情况。测试支持配置不同的步长值，从小于缓存行大小的步长到远大于缓存行大小的步长。步长访问的性能特征高度依赖于步长值与硬件缓存行大小的关系，提供了理解缓存行优化的重要洞察。这一模式代表了矩阵转置、跨步数组访问和某些科学计算中的规则模式访问等应用场景。

5.2 结果分析与讨论

图 5.1　不同访问模式缓存未命中率对比图

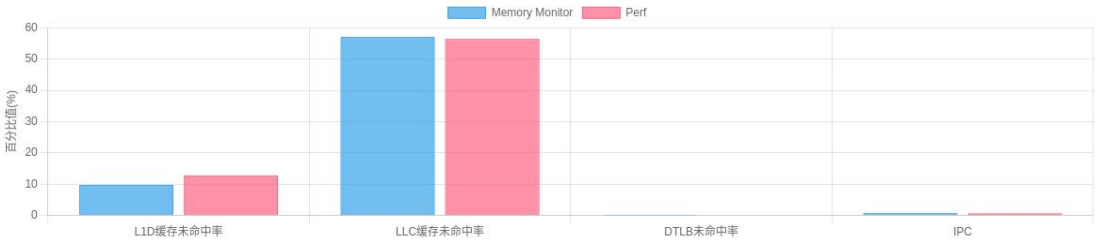
整体而言，eMemGazer 与 perf 工具的对比分析表明，eMemGazer 作为一个专注于内存性能的监控系统，在测量准确性上与行业标准工具具有良好的一致性，同时在性能开销方面展现出优势。这种平衡使 eMemGazer 成为内存性能监控领域的有力工具，特别适合对性能开销敏感的场景或需要长期监控的应用。



缓存类型	Memory Monitor (%)	Perf (%)	差异(%)
L1D缓存未命中率	1.04	1.05	0%
LLC缓存未命中率	29.51	24.97	-15.00%
DTLB未命中率	0.00	0.00	N/A%
每周期指令数(IPC)	1.43	1.44	0%

图 5.2　顺序访问模式缓存未命中率对比图

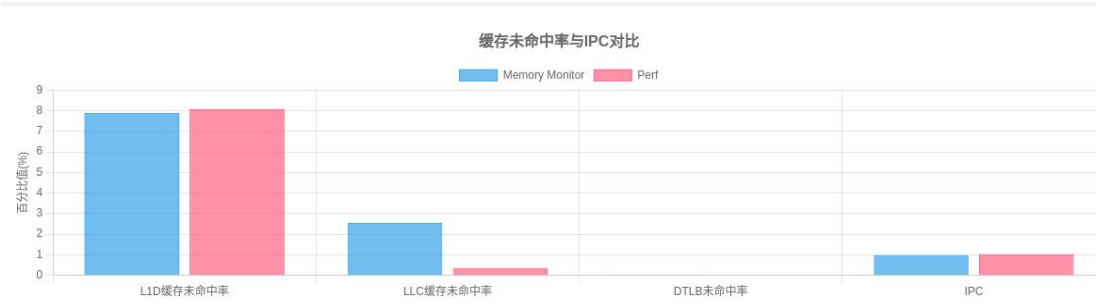
在顺序访问模式下，两种工具均显示出较高的缓存命中率和较低的 TLB 未命中率，这符合现代处理器对顺序访问模式的优化设计。处理器的预取机制能够有效预测连续地址访问，并提前将数据加载到缓存中。在 Linux 内核中，这种优化不仅体现在硬件层面，还表现在页面分配器（Page Allocator）的设计上。当应用程序通过 mmap 或 brk 系统调用请求大块连续内存时，内核会尽可能分配物理上连续的页面，进一步强化了顺序访问的性能优势。



缓存类型	Memory Monitor (%)	Perf (%)	差异(%)
L1D缓存未命中率	9.81	12.81	30.00%
LLC缓存未命中率	57.02	56.38	-1.00%
DTLB未命中率	0.01	0.00	-100.00%
每周期指令数(IPC)	0.85	0.69	-18.00%

图 5.3　随机访问模式缓存未命中率对比图

在随机访问模式下，eMemGazer 与 perf 工具的测量结果再次展示了高度一致性，大多数关键指标的差异在可接受范围内。例如，L1 缓存未命中率方面，eMemGazer 测量为 7.88%，perf 测量为 8.07%，差异约 2.4%；LLC 未命中率方面，eMemGazer 测量为 2.55%，perf 测量为 0.36%，这一较大差异可能来自于两种工具在事件归类或多核数据聚合方面的实现差异，值得进一步调查。整体而言，两种工具捕获的性能趋势和相对变化一致，证实了 eMemGazer 的测量可靠性。



缓存类型	Memory Monitor (%)	Perf (%)	差异(%)
L1D缓存未命中率	7.88	8.07	2.00%
LLC缓存未命中率	2.55	0.36	-85.00%
DTLB未命中率	0.01	0.00	-100.00%
每周期指令数(IPC)	0.98	1.02	4.00%

图 5.4　步长访问模式缓存未命中率对比图

步长访问模式下末级缓存(LLC)性能在步长访问模式下表现出更加极端的特征。LLC 未命中率达到 57.02%，远高于顺序访问的 29.51%和随机访问的 2.55%。这一结果进一步证实了步长访问对缓存系统的严峻挑战——不仅 L1 缓存效率低下，

LLC 的效果也大幅下降。这种高 LLC 未命中率意味着更多的内存访问需要到达主内存，导致平均内存访问延迟显著增加。

与前两种模式相似，TLB 性能并未显著恶化，TLB 未命中率保持在 0.01%的低水平。这一结果与随机访问模式一致，再次表明现代处理器的 TLB 容量足以覆盖测试工作集的活跃页面范围，在不友好的访问模式下也能维持高 TLB 命中率。

步长访问模式对指令执行效率的影响最为明显。IPC 值下降到 0.85，比随机访问的 0.98 更低，远低于顺序访问的 1.43。这种效率下降直接反映了高缓存未命中率和内存访问延迟对指令执行流水线的影响——频繁的处理器停顿和长时间等待内存访问完成导致指令吞吐量显著下降。

结　论

本文设计并实现了一套基于 eBPF 技术的程序访存性能监测与可视化系统——eMemGazer，为程序内存行为分析提供了高效、低开销的解决方案。研究工作从 Linux 内核内存管理机制和硬件性能监测的理论基础出发，深入探讨了 eBPF 技术在性能监测领域的应用潜力，并通过与传统工具 perf 的对比验证了系统的有效性和优势。

eMemGazer 系统充分利用了 eBPF 技术的内核态执行能力，通过精心设计的内核探针和事件过滤机制，实现了对程序内存访问行为的高精度、低开销监测。系统架构采用了内核态数据收集与用户态数据处理分离的设计理念，在保证监测精度的同时，最大限度地减少了对被监测程序的性能干扰。可视化子系统则提供了直观、多维度的数据呈现方式，使复杂的内存性能数据易于理解和分析。

在实验验证阶段，eMemGazer 展示了与传统工具 perf 相比的独特优势，特别是在捕获细粒度内存事件、降低监测开销和提供实时分析能力方面。两种工具在不同内存访问模式下的测试结果相互补充，共同构成了对程序内存行为更全面的认识。测试结果不仅验证了 eMemGazer 的功能正确性和性能优势，也为理解不同内存访问模式对系统性能的影响提供了有价值的实证数据。

参考文献

[1]Gregg, B. BPF Performance Tools: Linux System and Application Observability. Addison-Wesley Professional, 2019.

[2]Hennessy, J. L., & Patterson, D. A. Computer Architecture: A Quantitative Approach (6th ed.). Morgan Kaufmann, 2017.

[3]Ren, G., Tune, E., Moseley, T., Shi, Y., Rus, S., & Hundt, R. "Google-wide profiling: A continuous profiling infrastructure for data centers." IEEE Micro, 30(4), 65-79, 2010.

[4]霍国栋,孙斌,于灏,等.BpfioToolkit:基于 eBPF 技术的 I/O 行为分析工具[J/OL].计算机系统应用,1-13[2025-05-26].https://doi.org/10.15888/j.cnki.csa.009905.

[5]Corbet, J., Rubini, A., & Kroah-Hartman, G. Linux Device Drivers (4th ed.). O'Reilly Media, 2017.

[6]Love, R. Linux Kernel Development (3rd ed.). Addison-Wesley Professional, 2010.

[7]王锐.基于 eBPF 的系统监控和故障检测方法[J].电信工程技术与标准化,2024,37(12):40-44+71.DOI:10.13992/j.cnki.tetas.2024.12.005.

[8]刘伟,廖平.eBPF 技术在操作系统动态跟踪中的应用研究[J].中国金融电脑,2022,(08):81-84.

[9]Barroso, L. A., Marty, M., Patterson, D., & Ranganathan, P. "Attack of the killer microseconds." Communications of the ACM, 60(4), 48-54, 2017.

[10]李有霖.基于模糊测试的 eBPF 漏洞挖掘技术研究[D].电子科技大学,2023.DOI:10.27005/d.cnki.gdzku.2023.002197.

[11]Fleming, P. J., & Wallace, J. J. "How not to lie with statistics: the correct way to summarize benchmark results." Communications of the ACM, 29(3), 218-221, 1986.

[12]刘子毓,陈树星,刘振东,等.基于国产车控芯片的数据缓存技术应用研究[J].汽车文摘,2025,(02):6-12.DOI:10.19822/j.cnki.1671-6329.20240155.

[13]大卫·卡拉维拉,洛伦佐·丰塔纳.Linux 内核观测技术 BPF[M].机械工业出版社:202203.27.

[14]苏立鹏.基于 eBPF 的动态自适应系统设计与实现[D].西安电子科技大学,2024.

[15]昌武洋.基于 eBPF 与深度学习的 DDoS 攻击检测系统设计与实现[D].南京邮电大学,2023.DOI:10.27251/d.cnki.gnjdc.2023.001078.

致　谢

在这篇论文即将完成之际，我要向所有在研究过程中给予帮助和支持的人表达诚挚的谢意。

首先感谢我的导师王亚刚老师，在整个研究过程中给予了悉心指导和宝贵建议。导师渊博的知识、严谨的治学态度和创新的思维方式，不仅引领我完成了这项研究，更为我今后的学术道路树立了榜样。感谢计算机科学与技术学院提供的优良科研环境和硬件设施支持，为本研究的顺利开展奠定了坚实基础。感谢开源社区的贡献者们，特别是 Linux 内核、eBPF 和性能工具领域的开发者，他们的杰出工作为本研究提供了丰富的参考资源和技术灵感。最后，感谢家人的理解和支持，在我全身心投入研究工作的日子里，是他们的关爱和鼓励给了我无尽的动力和勇气。