Extrakce událostí ze souborových systémů

Návrh a implementace distribuované architektury

Technická zpráva FIT VUT v Brně

Ing. Radek Burget, Ph.D.
RNDr. Marek Rychlý, Ph.D.

Technická zpráva č. FIT-TR-2020-08
Fakulta informačních technologií, Vysoké učení technické v Brně

Last modified: 5. července 2020

Extrakce událostí ze souborových systémů
Návrh a implementace distribuované architektury

Ing. Radek Burget, Ph.D.
RNDr. Marek Rychlý, Ph.D.

Vysoké učení technické v Brně, email: {burgetr,rychly}@fit.vutbr.cz

Abstrakt Extrakce událostí s časovými razítky z lokálních souborových systémů představuje klíčový krok pro rekonstrukci časové osy, která vypovídá o využití daného zařízení (počítače, USB disku, apod.) v minulosti. Existující nástroje pro extrakci událostí jsou bez výjimky určeny pro spuštění na jednom lokálním počítači. Vzhledem k dostupné kapacitě současných pevných disků a dalších úložných zařízení je pak analýza obsažených souborových systémů časově velmi náročná. V tomto dokumentu proto popisujeme návrh, implementaci a experimentální ověření nového, distribuovaného řešení, které umožňuje rozložit proces extrakce událostí na velké množství výpočetních uzlů. To umožňuje zvýšit efektivitu celého procesu a současně integrovat získané výsledky s daty z dalších zdrojů do společné časové osy.

1 Úvod

Rekonstrukce časové osy představuje důležitý krok forenzní analýzy datových zdrojů. Jedná se o sběr událostí, o nichž jsou k dispozici informace v daném datovém zdroji a kterým je možno přiřadit konkrétní časový údaj. Nejčastějším datovým zdrojem jsou lokální souborové systémy dostupné např. na analyzovaných USB discích nebo na pevných discích počítačů [6,11], lze však uvažovat i online zdroje, jako např. účty na sociálních sítích [5]. Seřazením těchto událostí do časové posloupnosti lze získat časovou osu, která vypovídá o používání příslušného zařízení nebo účtu na sociální síti v minulosti [9].

V našem předchozím výzkumu v rámci projektu Tarzan jsme se zabývali rekonstrukcí časové osy zejména v souvislosti s využíváním účtů na sociálních sítích [1,2]. Navrhli jsme grafový datový model pro jednotnou reprezentaci událostí pocházejících z různých zdrojů a infrastrukturu umožňující extrakci informací o událostech, jejich další automatizovanou analýzu a interaktivní procházení uživatelem. Současně jsme se zabývali analýzou souvislostí mezi událostmi na účtech sociálních sítí a v lokálních souborových systémech. Zatímco pro získání informací z online zdrojů, ukládání a analýzu dat jsme navrhli distribuovanou architekturu založenou na platformě Tarzan [10], která umožňuje rozložit tyto činnosti efektivně na velké množství výpočetních uzlů, pro analýzu lokálních souborových systémů není takové řešení dosud k dispozici.

Události zaznamenané v souborových systémech lze podle [6] rozdělit na nízkoúrovňové (*low-level*), které lze přímo získat z analýzy souborového systému, jako např. vytvoření nebo změna souboru, změna konfigurační položky operačního systému (registry) nebo spuštění programu a vysokoúrovňové (*high-level*), které lze odvodit dodatečnou analýzou nízkoúrovňových událostí, jako např. připojení USB disku k počítači. Pro extrakci nízkoúrovňových událostí ze zařízení (např. z pevného disku) je nutné projít celý souborový systém a analyzovat každý jednotlivý soubor v závislosti na jeho typu. K tomuto účelu bylo vytvořeno větší množství nástrojů, jako např. plaso [4], TEAR [8,12], PyDFT [6], CyberForensic TimeLab [9] a další. Vzhledem k množství souborů uložených na běžném počítači představuje však analýza těmito prostředky časově i prostorově velmi náročnou úlohu, kdy extrakce událostí z běžného souborového systému může trvat i desítky hodin.

V této technické zprávě se proto věnujeme návrhu a implementaci distribuovaného řešení, které (podobně jako ve výše zmíněném případě analýzy online zdrojů) umožní rozložit celou analýzu na větší množství výpočetních uzlů s využitím platformy Tarzan a tím výrazně snížit časovou náročnost celého procesu. Popisované řešení je založeno na adaptaci existujícího, volně dostupného nástroje *plaso* pro distribuované prostředí, což zahrnuje jak jeho adaptaci na dříve navržený distribuovaný datový model, tak i návrh zcela nové infrastruktury pro spouštění jednotlivých částí extrakce. Současně vytvořené řešení umožňuje integrovat extrahované události ze souborových systémů s událostmi z online zdrojů do společné časové osy.

Struktura této technické zprávy je následující: V kapitole 2 představujeme nástroj plaso a jeho architekturu a v kapitole 3 rozebíráme použité technologie pro implementaci distribuovaného řešení. Kapitola 4 je věnována návrhu vlastního distribuovaného řešení, kapitola 5 popisuje implementační detaily a konečně kapitola 6 obsahuje detaily experimentálního vyhodnocení vytvořeného řešení a jeho výsledky.

1.1 Využití výsledků extrakce událostí

Vytvořené řešení poskytuje stejnou funkčnost, jako výše zmíněné existující nástroje pro extrakci událostí. Jeho základním použitím je tedy extrakce nízkoúrovňových událostí ze souborových systémů pro účely další analýzy, ať již automatizované, nebo ruční s cílem mapovat aktivitu na daném zařízení v minulosti. Základní případy použití tedy mohou zahrnovat např.

- Zjištění historie konkrétních souborů – kdy byl soubor vytvořen, jakým způsobem se dostal na dané zařízení (stažení z webu, kopírování z přenosného média apod.), případně zda byl soubor upravován pomocí konkrétní aplikace.
- Zjištění aktivity uživatelů – spuštění počítače, přihlášení a odhlášení ze systému, operace s webovým prohlížečem (návštěvy stránek, stahování souborů) apod.

V těchto základních scénářích je přínosem popisovaného nového řešení zejména vyšší efektivita extrakce událostí daná možností distribuce celého procesu na více výpočetních uzlů.

Dalším přínosem je pak použití jednotného datového modelu pro popis událostí, který umožňuje jednak využití dříve vytvořených interaktivních nástrojů pro ruční zkoumání posloupnosti událostí a dále integraci s dalšími zdroji událostí, zejména ze sociálních sítí. To umožňuje využít získaná data k nalezení odpovědí na další otázky ohledně původů souborů [2]:

- Byl daný lokální soubor stažen ze sociální sítě? Pokud ano, z jakého konkrétního zdroje? (např. profilu)
- Jaká je historie tohoto souboru v rámci sledovaných profilů na sociálních sítích? Kdy byl soubor publikován a kým?
- Existuje souvislost (obsahová, časová, apod.) mezi aktivitou lokálního počítače a některých profilů na sociálních sítích? Lze z toho usuzovat, že např. uživatel počítače je současně vlastníkem daných profilů?

Zvolený grafový model založený na standardu RDF rovněž otevírá další možnosti pro analýzu uložených dat např. pomocí dotazovacího jazyka SPARQL, pravidel zapsaných v jazyce SWRL a dalších prostředků.

2 Nástroj Plaso

Plaso[1] je obecný nástroj pro extrakci časových událostí z různých souborů ve zkoumaném souborovém systému a jejich agregaci. Je schopen projít souborový systém extrahovat události z více než 100 podporovaných souborových a databázových formátů. Kromě systému Windows podporuje i další běžné systémy: Linux, Mac OS X a Android. Vzhledem k tomu, že nástroj plaso je v současnosti (červen 2020) aktivně vyvíjen jako software s otevřeným kódem, jedná se o velmi populární řešení, které může být použito jako zdroj dat pro další analýzu softwarovými prostředky [3].

Samotný nástroj Plaso představuje jádro celého řešení (*backend*), které implementuje vlastní analýzu souborového systému. Výsledkem této analýzy jsou údaje obvykle o velkém množství nalezených událostí, které jsou uloženy ve speciálním datovém souboru. Pro přípravu dat, řízení extrakce událostí a zpracování výsledků jsou k dispozici následující doplňkové nástroje:

- `image_export` – nástroj pro export obsahu ze zdrojového souborového systému na základě různých kritérií
- `log2timeline` – spuštění vlastní extrakce nad zdrojovými daty a uložení nalezených událostí
- `pinfo` a `psort` – nástroje pro filtrování a zobrazení nalezených událostí v různých formátech a export výsledků.

[1] https://github.com/log2timeline/plaso

Řešení je implementováno v jazyce Python a typicky se ovládá z příkazové řádky pomocí uvedených nástrojů. Informace o nalezených událostech lze exportovat do celé řady výstupních formátů, jako např. JSON (zejména pro další počítačové zpracování) nebo Microsoft Excel (pro ruční analýzu).

2.1 Architektura nástroje

Zpracování vstupního souborového systému nástrojem Plaso se skládá z následujících dílčích úloh:

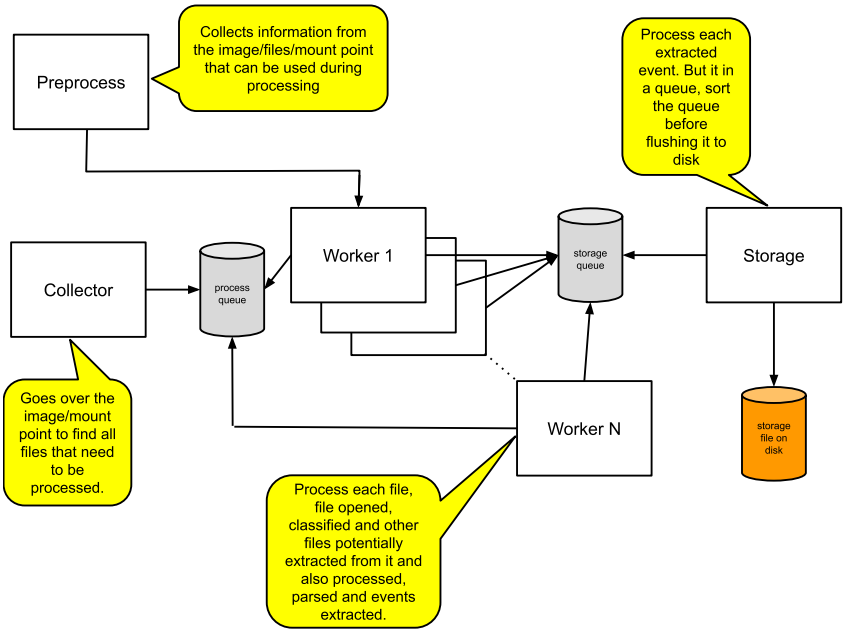
- **Předzpracování** (*preprocessing*) – získání základních údajů o vstupním souborovém systému, které zahrnují typ operačního systému, časové zóny, jméno počítače, jména uživatelů a cesty k jejich domácím adresářům a další globální informace.
- **Výběr vstupů** (*collection*) – získání seznamu vstupních souborů, ze kterých se budou extrahovat informace, v nejjednodušším případě rekurzivním průchodem adresářové struktury souborového systému, případně s využitím různých filtrů.
- **Extrakce událostí** (*extraction*) – získání časových událostí, tedy klíčová část celé funkcionality. Pro každý vstupní soubor je klasifikován jeho typ a následně jsou aplikovány speciální extraktory příslušné k určenému typu souboru, které analyzují jeho obsah a produkují posloupnost událostí.
- **Uložení výsledků** (*storage*) – ukládání informací o nalezených událostech do interní vyrovnávací paměti a následně do výsledného datového souboru.

Základní architektura nástroje, která pokrývá výše uvedené činnosti, je patrná z obrázku 1. Klíčová úloha extrakce událostí je implementována formou oddělených pracovních položek (*workers*), které mohou být v případě potřeby spuštěny paralelně ve více vláknech. Architektura celého nástroje však počítá pouze s paralelním spuštěním v rámci jednoho výpočetního uzlu bez možnosti distribuovaného zpracování na více počítačích.

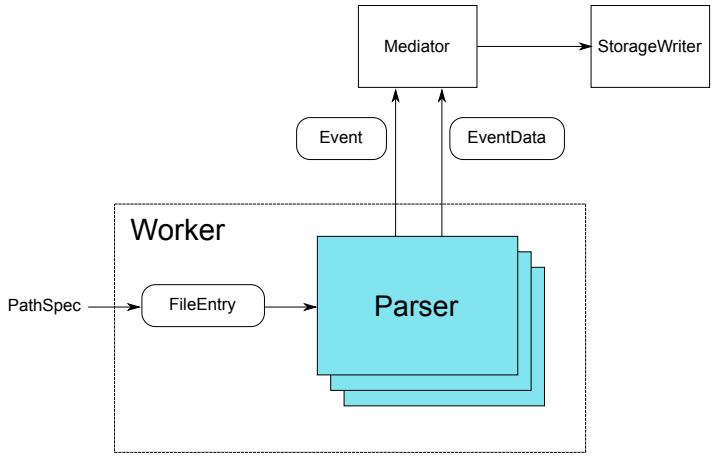
Obrázek 2 podrobněji ukazuje strukturu jednotlivých pracovních položek. Každý zpracovávaný soubor (*file entry*) je zpracován jedním nebo více *parsery* (někdy též nazývané extraktory). Jedná se o procedury specializované na konkrétní typy souborů, které implementují dekódování vstupního souboru a získání informací o událostech specifických pro jeho typ. Nástroj Plaso aktuálně obsahuje desítky parserů pro dekódování různých souborů od spustitelných přes datové soubory různých databází až po konfigurační soubory různých operačních systémů.

Výsledkem běhu parseru jsou datové struktury popisující nalezené události. Výsledky všech spuštěných parserů sbírá tzv. *Mediator* a zajišťuje jejich uložení prostřednictvím objektu *StorageWriter*.

Jak bylo naznačeno výše, z pohledu paralelizace celého procesu je současná architektura nástroje Plaso navržena tak, že umožňuje paralelní běh jednotlivých parserů v samostatných vláknech. To umožňuje efektivní využití současných počítačů, které disponují několika procesorovými jádry schopnými realizovat paralelní běh těchto procesů. Současně však celá architektura využívá sdílených



Obrázek 1. Architektura nástroje Plaso [13].



Obrázek 2. Extrakce událostí v nástroji Plaso pomocí parserů.

datových struktur uložených v paměti, která je dostupná všem procesům. Jedná se jednak o informace o vstupních souborech a zejména pak o sběr výsledků extrakce událostí využívající sdílený *Mediator*. To znemožňuje přímé na nasazení celého nástroje v distribuované architektuře využívající více fyzických výpočetních uzlů, kde tyto sdílené datové struktury nelze použít.

V dalších kapitolách proto popisujeme nově vytvořené řešení, které využívá pouze jednotlivých parserů nástroje Plaso, kolem kterých je vybudována zcela nová infrastruktura zajišťující běh na větším množství výpočetních uzlů a distribuci zdrojových i cílových dat mezi těmito uzly.

3 Související technologie

Navržené řešení využívá integrovanou platformu vyvinutou v rámci projektu Tarzan v předchozích letech [10]. Tato platforma integruje celou řadu technologií pro distribuované zpracování dat, z nichž nejdůležitější z pohledu extrakce událostí ze souborových systémů jsou tyto:

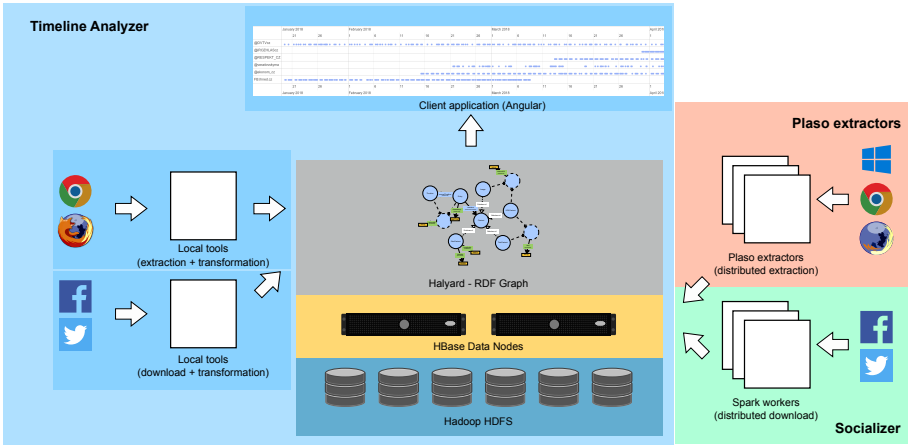
- **Apache Hadoop** – zejména distribuovaný souborový systém HDFS [14], který je využit pro zpřístupnění zdrojových dat (souborů analyzovaného souborového systému) a současně i pro ukládání výsledků extrakce prostřednictvím sloupcové databáze HBase, jak popisujeme dále.
- **Apache Spark** – technologie pro řízení distribuci úloh napříč výpočetními uzly využitá pro distribuované spouštění jednotlivých úloh na větším množství fyzických uzlů.

Integrace extraktorů nástroje Plaso popsaných v předchozí kapitole s touto distribuovanou platformou představuje poměrně složitý technický problém, jehož řešení je podrobně popsáno v kapitole 4.

Současně navržené řešení navazuje na předchozí výzkum v rámci projektu Tarzan, který se zaměřoval na analýzu událostí zejména na sociálních sítích. V rámci tohoto výzkumu byl vytvořen nástroj *TimelineAnalyzer* [2], který umožňuje integrovat různé zdroje událostí a sledovat vzájemné vazby mezi nalezenými událostmi. Z tohoto pohledu lze extrakci událostí ze souborových systémů chápat jako další zdroj informací o událostech, který je možno integrovat do již existujícího řešení.

Nástroj TimelineAnalyzer využívá společné distribuované úložiště informací o událostech, které je založené na grafovém datovém modelu RDF [7]. Informace o událostech ze všech zdrojů (jak sociálních sítí, tak souborových systémů) jsou reprezentovány ve formě grafu, který umožňuje efektivně sledovat jednotlivé události, zdroje, kterých se události týkají a podobně. Tento graf je v rámci distribuované platformy uložen do speciální databáze postavené na následujících technologiích:

- **Souborový systém HDFS** – slouží jako nejnižší vrstva úložiště, jak bylo zmíněno výše. Zajišťuje uložení datových souborů, které však nejsou využívány přímo, nýbrž prostřednictvím databázového řešení HBase.



Obrázek 3. Integrace nástroje Plaso extractors do infrastruktury TimelineAnalyzer.

– **Apache HBase** – škálovatelná sloupcová (NoSQL) databáze zajišťující organizaci dat do tabulek a jejich uložení na HDFS.
– **Halyard** – úložiště grafových dat RDF zajišťující transformaci grafové reprezentace a uložení do tabulek HBase.

Pro integraci nástroje pro distribuovanou extrakci událostí ze souborových systémů do této existující infrastruktury jsou proto nutné následující kroky:

– Zajištění transformace extrahovaných dat o událostech do datového modelu RDF kompatibilního s nástrojem TimelineAnalyzer. Tento navržený model je reprezentován pomocí ontologií a byl detailně popsán v předchozí technické zprávě [2].
– Export takto popsaných dat do sdíleného úložiště Halyard, které běží jako součást distribuované platformy.

Obrázek 3 ukazuje architekturu existujícího nástroje TimelineAnalyzer a integraci nástroje pro distribuovanou extrakci událostí ze souborových systémů (Plaso extractors). Výsledné řešení umožňuje analýzu získaných informací o událostech v souborovém systému společně s událostmi z ostatních zdrojů (sociálních sítí) a využívat webové rozhraní TimelineAnalyzer pro interaktivní procházení těchto událostí.

4 Architektura distribuovaného řešení

Zpracování vstupního souborového systému v nástroji Plaso probíhá jako sekvence úloh: předzpracování, výběr vstupů, extrakce událostí a uložení výsledků,

jak bylo popsáno v kap. 2. Protože souborový systém obsahuje obvykle více souborů a pro zpracování každého z nich je možno použít více extraktorů, je uvedená sekvence úloh vykonávána několikrát, přičemž jednotlivé průběhy jsou na sobě typicky nezávislé.

Jedná se tedy o výpočetní úlohu, kterou lze snadno provádět paralelně a distribuovaně (jde o tzv. „an embarrassingly parallel problem“). Lze rozdělit zpracování souborového systému na několik paralelních úloh, které je možno provádět na několika výpočetních uzlech. Zároveň se jedná o dávkové zpracování velkých dat, kdy je celý souborový systém (všechny jeho soubory) zpracován naráz v jedné dávce a kdy zpracování vyprodukuje jeden soubor výsledků (jednu množinu extrahovaných událostí).

Výše uvedené skutečnosti pak určují architekturu navrženého řešení, jehož komponenty jsou popsány v následujících podkapitolách.

4.1 Distribuované úložiště vstupních dat

Pro distribuované zpracování dat je vhodné mít tato data uložena v distribuovaném úložišti. Distribuované uložení dat pak umožní rozdělit výpočet tak, aby byly jednotlivé výpočetní úlohy prováděny co nejblíže místu uložení jejich vstupních dat (tzv. „data locality“).

Pro distribuované uložení velkých dat jsou k dispozici dvě možnosti – distribuovaná (NoSQL) databáze nebo distribuovaný souborových systém. Protože v našem případě zpracováváme obsahu souborového systému, tedy velkou kolekci binárních různorodých dat (souborů), a tato data budou čtena v dávce pouze jednou, je vhodné použít distribuovaný souborový systém.

Jako distribuované úložiště byl zvolen Hadoop File-system (HDFS) z projektu *Apache Hadoop*[2]. Tato technologie je de-facto standardem v oblasti distribuovaného uložení a zpracování velkých dat (Big data) a je navržena pro sekvenční (tedy jednorázové) čtení binárních dat. Navíc je možno umístit jednotlivé soubory vstupních dat (tj. souborového systému na vstupu) přímo jako soubory HDFS a při extrakci k nim přistupovat jednotlivě. Metadata těchto souborů jsou při nahrání vstupních dat (obrazu souborového systému na vstupu) v rámci předzpracování extrahována a uložena na HDFS společně s datovými soubory (metadata obsahují např. uživatele a skupiny, jejich oprávnění, časové značky, uživatelská metadata, a.j., dle možností vstupního souborového systému).

4.2 Paralelní a distribuované zpracování – extrakce událostí

Při zpracování vstupních dat (souborů vstupního souborového systému) jsou jednotkami zpracování jednotlivé soubory. Pro každý z nich probíhá v rámci zpracování následující aktivity (analogicky k průběhu zpracování popsanému v kap. 2):

– **Analýza pro výběr extraktorů** – Každá jednotlivá úloha vezme jeden soubor na vstupu a tento nejprve analyzuje a aplikuje filtry, na základě

[2] https://hadoop.apache.org/

kterých se zvolí zda a jak bude soubor dále zpracován. Filtruje se na základě názvu souboru a jeho metadat, která byla získána v rámci předzpracování (v rámci nahrání souboru ze vstupního souborového systému na HDFS). Výsledkem tohoto kroku je množina jmen extraktorů, které se použijí v kroku následujícím.
- **Extrakce událostí** – Extraktory zvolené v předchozím kroku jsou spuštěny a to paralelně (souběžně všechny naráz) a distribuovaně na více výpočetních uzlech. V tomto kroku se využívají extraktory z projektu Plaso (viz kap. 2), jejichž zdrojový kód je z projektu Plaso převzat, vyčištěn (izolován od ostatních komponent projektu Plaso), zapouzdřen za navržené rozhraní a spouštěn na cílových výpočetních uzlech.
- **Sběr a uložení výsledků** – Události vyprodukované paralelně a distribuovaně spuštěnými extraktory je potřeba sesbírat a nahrát do úložiště výsledků. Z pohledu distribuovaného zpracování dat se jedná o úlohu agregace.

Z hlediska distribuovaného zpracování dat se jedná o tzv. úlohu „map-reduce“, tedy rozdělení vstupních dat na menší části (vstupní soubory), jejich jednotlivé zpracování stejným algoritmem (fáze „map“) a finální sběr a agregace výsledků jednotlivých zpracování (fáze „reduce“).

Byla zvolena technologie Apache Spark[3], což vysoce optimalizovaný rámec pro map-reduce úlohy běžící nad Apache Hadoop (viz kap. 4.1). Oproti implementaci map-reduce přímo v projektu Hadoop, řešení Spark je rychlejší a vhodnější pro menší úlohy. Další výhodou je podpora programovacího jazyka Python, takže je možno integrovat extraktory z projektu Plaso, které jsou implementovány rovněž v jazyce Python.

4.3 Úložiště extrahovaných událostí

Extraktory z předchozí části generují záznamy o událostech ze zpracovávaných vstupních souborů. Každá událost je popsána časovou značkou (čas, kdy v souboru původně vznikla; např. čas přístupu k webové stránce v souboru historie webového prohlížeče) a atributy s jejich hodnotami, které událost popisují (např. URL webové stránky, klíčová slova vyhledávače, atp.). Atributy událostí jsou strukturované, ale různé v závislosti na druhu události (různé extraktory produkují různé druhy událostí).

Z výše uvedeného popisu extrahovaných událostí plyne, že je vhodné použít pro jejich uložení databázi. A protože se jedná o výstupy distribuovaného zpracování, je vhodné použít distribuovanou databázi (opět princip tzv. „data locality“, kdy je vhodné zapsat data co nejblíže uzlu, který tato data produkuje; viz také kap. 4.1).

Pro distribuované uložení extrahovaných událostí byla zvolena NoSQL databáze Apache HBase[4], která využívá rámec Apache Hadoop a jeho souborový systém HDFS. Díky tomu je možné uložit události do stejného HDFS systému, který je již použit pro uložení vstupních dat (viz kap. 4.1).

[3] https://spark.apache.org/

[4] https://hbase.apache.org/

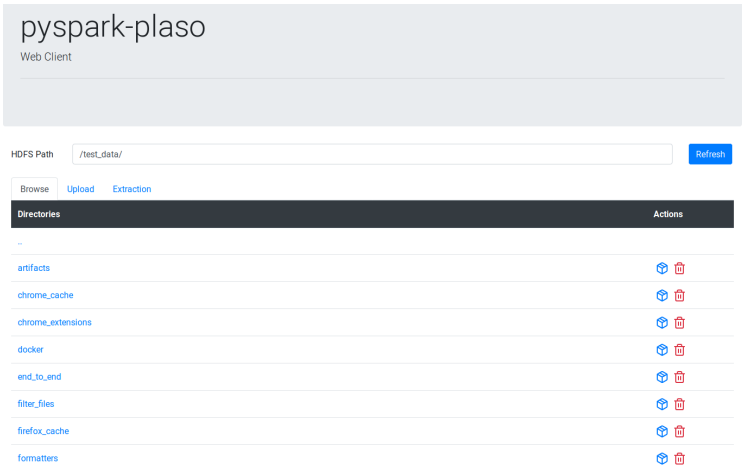
Protože se bude s extrahovanými událostmi dále pracovat, zejména hledat jejich souvislosti (viz také kap. 6), je použita nadstavba databáze HBase v podobě projektu Halyard[5], což je RDF grafová databáze pro uložení sémantických dat s dotazovacím jazykem SPARQL. Následná analýza s využitím databáze Halyard probíhá v nástroji Timeline Analyzer[6], který je rovněž produktem projektu TARZAN a byl popsán v samostatné technické zprávě [2].

4.4 Aplikační a uživatelské rozhraní

Celý proces zpracování a jeho jednotlivé kroky v podobě nahrání vstupních dat do HDFS, vlastní extrakce (tj. analýza a spuštění extraktorů) a uložení výsledných událostí, lze ovládat z aplikačního a uživatelského rozhraní.

Aplikační rozhraní (API) je vhodné pro použití dalšími (klientskými) aplikacemi, např. pro integraci do ostatních aplikací či platformy TARZAN. Toto rozhraní je dostupné pomocí protokolu HTTP v podobě REST (Representational State Transfer) architektury, kde jednotlivé zdroje REST jsou jednotlivé aktivity (např. nahrání, extrakce, aj.) a cesty (adresáře či soubory) vstupních dat. Konkrétní URL s metodami, parametry, vstupy a výstupy, jsou popsány v kap. 5.

Pro snadné ovládání je k dispozici také uživatelské rozhraní dostupné pomocí webového prohlížeče (viz obrázek 4). Uživatelské rozhraní umožňuje provádět stejné akce, jako jsou k dispozici ve dříve popsaném REST API, tj. nahrát vstupní data na HDFS, provést extrakci a uložit či získat výsledek.



Obrázek 4. Grafické uživatelské rozhraní nástroje PySpakr Plaso.

[5] https://merck.github.io/Halyard/
[6] https://github.com/nesfit/timeline-analyzer

5 Implementace

Implementace řešení pro distribuovanou extrakci ze souborových systému se sestávala z následujících kroků:

1. *Konfigurace a nasazení jednotlivých služeb* (serverů) technologií HDFS (distribuovaný souborový systém z projektu Apache Hadoop), Apache Spark (distribuovaný výpočetní systém s úlohami v jazycích Java a Python), Apache HBase (distribuovaná databáze postavená na HDFS), Apache Zookeeper (synchronizační služba pro HBase), Halyard (grafové RDF databáze postavená na HBase),
2. *Návrh a implementace vlastních komponent systému*, tj. PySpark Plaso (řídí proces analýzy vstupů a extrakce událostí nad platformou Spark, používá jednotlivé extraktory z projektu Plaso, poskytuje webová a aplikační programové rozhraní pro nahrávání vstupů a ovládání extrakce), Timeline Analyzer (zobrazuje extrahované události jako RDF zdroje načtené z databáze Halyard),
3. *Integrace komponent a technologií a nasazení*: zapouzdření použitých technologií a komponent do Docker kontejnerů a samostatné nasazení jednotlivých Docker kontejnerů v rámci virtualizované infrastruktury pomocí technologie Docker-compose a Kubernetes.

5.1 Konfigurace použitých služeb a technologií

Protože bylo použito několik technologií postavených nad platformou Apache Hadoop (konkrétně HDFS, Spark a HBase), bylo potřeba zvolit jednotnou verzi a konfiguraci této platformy. V opačném případě by nebylo možno tyto technologie později integrovat (např. kvůli nekompatibilitě verzí použitých knihoven). Byla navržena základní konfigurace Apache Hadoop (tzv. „Hadoop Base“) a tato dále doplňována dle potřeby jednotlivých technologií.

Vzniklá základní konfigurace Hadoop byla využita i pro konfiguraci platformy Apache Spark, na které běží výpočetní úlohy. Protože úloha analýzy a extrakce implementovaná v jazyce Python potřebuje načítat data ze souborového systému HDFS, byla konfigurace doplněna o knihovnu PyArrow[7], která poskytuje ovladač pro tento souborový systém. Knihovnu bylo nutno slinkovat (napojit dynamickým linkerem) na sdílenou knihovnu „libhdfs.so“ z připravené konfigurace platformy Apache Hadoop.

Grafová RDF databáze Halyard byla napojena na distribuovanou databázi projektu Apache HBase využívající HDFS z připravené konfigurace Apache Hadoop. Zde bylo potřeba opět sjednotit verze použitých knihoven.

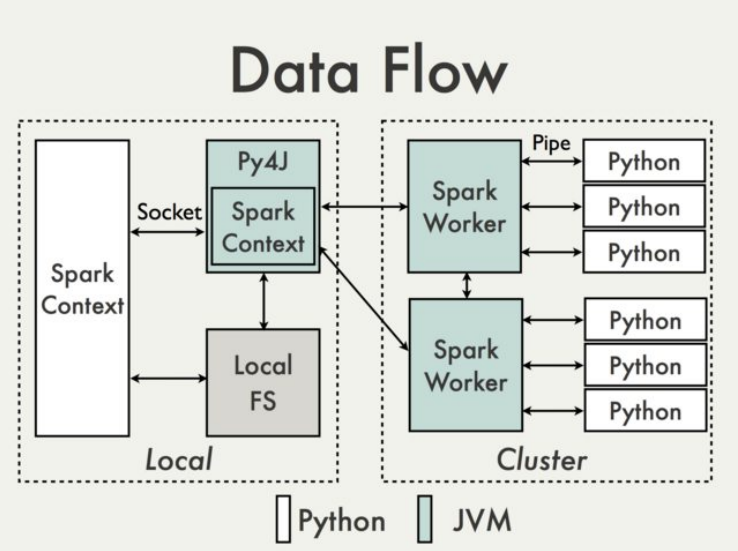
Nakonec byl analyzován původní systém Plaso (viz kap. 1) a připravenou prostředí pro izolovaný běh jednotlivých extraktorů z tohoto systému – Python prostředí se sadou knihoven a několika adaptéry oddělujícími extraktory z původní architektury systému Plaso. Připravené Python prostředí bylo integrováno do konfigurace platformy Spark tak, aby bylo možno spustit izolované extraktory jako PySpark distribuované úlohy (viz kap. 5.2).

[7] https://arrow.apache.org/docs/python/

5.2 Implementace komponenty PySpark Plaso

PySpark Plaso představuje hlavní implementovanou komponentu prezentovaného systému pro distribuovanou extrakci událostí ze souborových systémů. Tato komponenta implementuje a řídí proces analýzy vstupů a extrakce událostí a poskytuje také uživatelské (web) a programové rozhraní (REST API), jak bylo popsáno v kap. 4.2.

Analýza vstupů, extrakce událostí a uložení výsledků Protože platforma Spark je primárně určena pro programovací jazyk Java a Scala, kdy výsledné programy běží nad Java Virtual Machine (JVM), musí být klíčové části aplikace využívající Spark také v jazycích Java nebo Scala. V našem případě bylo potřeba použít extraktory z projektu Plaso, které jsou napsány v programovacím jazyce Python. Přestože Spark poskytuje podporu pro jazyk Python v podobě PySpark, jedná se spíše o okrajové řešení, kdy je vytvořený Python kód řídící distribuovaný výpočet spouštěný v prostředí Java pomocí Py4J, zatímco část Python kódu popisující vlastní distribuované úlohy je spouštěna přímo nativním Python interpretem. Obě části aplikace, tj. část běžící v JVM a část běžící v Python interpretu, si pak předávají řízení a data, jak je popsáno na obr. 5.



Obrázek 5. Tok dat mezi částmi Spark aplikace nad JVM a v Python (převzato z https://cwiki.apache.org/confluence/display/SPARK/PySpark+Internals).

Rozdělení aplikace na část běžící pod JVM a část prováděnou interpretem Python komplikovalo implementaci – bylo potřeba implementovat různé metody přístupu k HDFS (nativní přístup pomocí knihoven Hadoop dostupných na platformě Spark v jazyce Java vs. přístup z Python aplikace pomocí knihovny PyArrow) či navázat při provádění distribuovaných úloh běžících na jednotlivých výpočetních uzlech první část spuštěnou v interpretu Python (extraktory z projektu Plaso získávající události ze vstupů) na druhou část běžící nad JVM (uložení výsledných událostí do databáze Halyard přes příslušný ovladač dostupný jen v jazyce Java).

Uživatelské a implementační rozhraní Kromě implementace analýzy vstupů a extrakce událostí popsané v předchozí části bylo potřeba implementovat i rozhraní pro ovládání těchto procesů a to jak programové rozhraní (API) pro integraci s jinými komponentami platformy TARZAN, tak uživatelské rozhraní pro ovládání koncovými uživateli.

Programové rozhraní bylo navrženo jako REST API a implementováno v jazyce Python pomocí knihovny Flask[8]. Byly implementovány následující REST zdroje dostupné pod příslušnými URL (následující popis je uvozen HTTP metodou a URL pro přístup k REST zdroji (s případnými parametry)):

- `GET /api/ls/<path>` – vrací JSON dokument s výpisem adresářů a souborů uložených v HDFS pro zpracování pomocí PySpark Plaso (vstupní data případné extrakce),
- `GET,PUT,POST,DELETE /api/file/<path>` – umožňuje stáhnutí (GET), nahrání (PUT nebo POST), nebo smazání (DELETE) souboru z/do HDFS adresáře dané cesty,
- `GET,PUT,POST /api/zip/<path>` – umožňuje stáhnutí (GET) nebo nahrání (PUT nebo POST) souborů a adresářů zabalených v archivu ZIP z/do HDFS adresáře dané cesty,
- `GET,DELETE /api/rm/<path>` – umožňuje smazání souboru či adresáře dané cesty z HDFS,
- `POST /api/file-form/<path>` & `POST /api/zip-form/<path>` – umožňuje nahrání souboru (file-form) či obsahu ZIP archivu (zip-form) do HDFS adresáře dané cesty jako obsahu typu „multipart/form-data“, což se používá při odeslání souborů pomocí HTML formuláře,
- `GET /api/extract/<path>` – zpracuje obsah HDFS adresáře či souboru dané cesty a extrahuje z nich události, které pak vrátí v kolekci v JSON dokumentu,
- `GET /api/extract-to-halyard/<path>` – zpracuje obsah HDFS adresáře či souboru dané cesty a extrahuje z nich události, které uloží jako RDF zdroje do databáze Halyard pro další zpracování pomocí nástroje Timeline Analyzer.

[8] https://palletsprojects.com/p/flask/

5.3 Implementace komponenty Timeline Analyzer

V rámci řešení projektu byla pro analýzu událostí extrahovaných ze vstupních dat implementována webová aplikace Timeline Analyzer[9]. Její implementace byla popsána v samostatné technické zprávě [2].

5.4 Integrace služeb a technologií pomocí Docker

Implementace systému pro distribuovanou extrakci událostí ze souborových systému zahrnuje několik komponent, ať už se jedná o zkonfigurované služby (HDFS, Spark, HBase, Zookeeper, Halyard) nebo o vytvořené komponenty systému (PySpark Plaso a Timeline Analyzer). Každá z těchto komponent představuje alespoň jednu běžící aplikaci a tyto aplikace je nutno nasadit na uzlech infrastruktury a vhodně nastavit a propojit, což je netriviální úloha. Navíc, způsob nasazení závisí na vlastnostech infrastruktury, kde mají být technologie a komponenty nasazeny (počet uzlů, jejich vlastnosti a způsob jejich propojení, zabezpečení, případné dynamické rozdělení zátěže či schování služeb za proxy servery). Situaci komplikuje i fakt, že použité technologie a komponenty mnohdy potřebují speciální nastavení běhového prostředí (knihovny, proměnné prostředí, atp.), jak bylo popsáno v kap. 5.1.

Z výše uvedeného plyne, že je vhodné proces nasazení automatizovat a umožnit jeho přizpůsobení. Zde lze zvolit dvě strategie: použít nástroj pro automatizaci nasazení (např. Ansible[10] nebo Puppet[11]), nebo použít některou z technik virtualizace a nasazovat kontejnery s před-připravenými službami a komponentami.

V případě případě našeho systému byla zvolena kontejnerová virtualizace pomocí technologie Docker[12], kdy je každá služba či komponenta zapouzdřena do samostatného Docker kontejneru s definovaným rozhraním (síťové porty) a parametry (proměnné prostředí). Tyto kontejnery jsou pak nastaveny, nasazeny a propojeny pomocí technologií Docker compose[13] (distribuovaně pomocí Docker Swarm[14]) či Kubernetes[15]. Typické nasazení obsahuje následující uzly:

- 1 (či více v případě redundance) HDFS NameNode uzlů řídících distribuované úložiště vstupních dat,
- alespoň 1 (typicky více) HDFS DataNode uzlů pro vlastní distribuované uložení vstupních dat,
- 1 (či více v případě redundance) HBase Master uzlů řídících distribuovanou databázi extrahovaných událostí,

[9] https://github.com/nesfit/timeline-analyzer
[10] https://www.ansible.com/
[11] https://puppet.com/
[12] https://www.docker.com/
[13] https://docs.docker.com/compose/
[14] https://docs.docker.com/engine/swarm/
[15] https://kubernetes.io/

– alespoň 1 (typicky více) HBase RegionServer uzlů pro distribuované uložení databáze v HDFS,
– 1 (či více v případě redundance) ZooKeeper uzlů pro synchronizaci distribuované databáze,
– 1 (či více v případě dělení zátěže) Halyard WebApps uzel pro využití distribuované databáze HBase jako grafové RDF databáze a její dotazování v jazyce SPARQL,
– 1 (či více v případě redundance) Spark Master uzlů řídících distribuovaný výpočet,
– alespoň 1 (typicky více) Spark Worker uzlů provádějící vlastní distribuovaný výpočet,
– 1 (či více v případě dělení zátěže) PySpark Plaso uzel poskytující uživatelské a programové rozhraní pro správu vstupních dat v HDFS a jejich extrakci pomocí Plaso extraktorů spouštěných dle potřeby distribuovaně ve Spark (vč. uložení do HBase přes rozhraní Halyard),
– 1 (či více v případě dělení zátěže) Timeline Analyzer uzel poskytující uživatelské rozhraní pro analýzu extrahovaných událostí dotazováním Halyard nad daty v HBase.

Jedná se tedy o alespoň 10 nasazení Docker kontejnerů v minimálním případě. V tomto případě je možno umístit všechny nasazené kontejnery na jeden uzel (stroj) a získat tak lokální nasazení systému, např. na stolním počítači. Při škálování distribuovaného úložiště, výpočtu či databáze, při větší robustnosti řešení díky redundanci, anebo při rozdělení zátěže, může počet nasazených Docker kontejnerů výrazně narůst a tyto jsou nasazeny na více uzlech (strojích) clusteru.

Rozmístění konkrétních kontejnerů na uzly infrastruktury a jejich síťové propojení je popsáno v `docker-compose.yml` souboru (při použití technologie Docker-compose, např. pro nasazení na jedno uzlu, kde běží Docker, nebo na skupině uzlů spravovaných Docker Swarm) nebo v souborech s definicemi zdrojů pro Kubernetes (zejména pro nasazení na skupině uzlů). Soubory s ukázkovým nasazením jsou součástí vzniklého projektu PySpark Plaso[16].

6 Experimentální vyhodnocení

Komponeta PySpark Plaso (viz kap. 5.2) implementuje stejnou funkcionalitu, jako původní projekt Plaso (viz kap. 2), ale výpočet, tj. analýza vstupních souborů a extrakce událostí, běží distribuovaně nad platformou Spark (s distribuovaným úložištěm vstupních dat v HDFS). Pro vyhodnocení je tedy potřeba ověřit

– *funkčnost* – tedy, že nad stejnými vstupními daty naleznou PySpark Plaso a původní Plaso stejnou množinu událostí,
– *výkon* – tedy, že ve stejném prostředí, se stejnými nastaveními analýzy a extrakce a se stejnými vstupními daty bude výkon PySpark Plaso alespoň stejný (případně vyšší), než je výkon původního Plaso (výkon je měřen jako rychlost vygenerování kompletního seznamu událostí ze vstupních dat).

[16] https://github.com/nesfit/pyspark-plaso

6.1 Konfigurace prostředí pro experimenty

Původní Plaso není možno spustit distribuovaně na více výpočetních uzlech[17], a proto budou následující experimenty s nově vyvinutým PySpark Plaso omezeny na jeden výpočetní uzel (tedy se bude jednat o minimální nasazení, kdy jsou všechny kontejnery provozovány na jednom stroji; viz kap. 5.4).

Hardware Experimenty byly provedeny na stroji Supermicro SC113TQ-600 jména „gort.fit.vutbr.cz“, který disponuje 64bitovým procesorem Intel Xeon E5-2620 v3 na frekvenci 2,4 GHz s 6 jádry (12 vláken); 4×8 GB RAM s moduly Samsung M393A1G40DB0-CPB typu DIMM DDR4 na 2,133 GHz; diskem SSD 500 GB model CT500MX500SSD1 na rozhraní SATA3 s reálnou přenosovou rychlostí čtení 10 GB/s s cache a 450 MB/s bez cache meřeno pomocí nástroje hdparm[18].

Software Na stroji pro experimenty běžel operační systém s jádrem Linux 4.19 distribuce NixOS 19.09. Byly využívány technologie Apache Hadoop verze 3.1.3, Apache Spark verze 2.4.4, Apache HBase verze 2.2 a s ní distribuovaný Apache Zookeeper, Halyard verze 3.0 a sestavení PySpark Plaso aktuální verze. Původní Plaso bylo ve verzi 20200227.

6.2 Vstupní data a použité extraktory

Jako vstupní dat pro experimenty byla použita testovací data z distribuce projektu Plaso verze 202002027[19]. Testovací data obsahují 318 souborů ve 21 adresářích. V případě PySpark Plaso byla data nahrána na distribuovaný soborový systém HDFS a v případě původního Plaso byla k dispozici na disku lokálního souborového systému BTRFS. Oba souborové systémy, tj. distribuovaný HDFS a lokální BRTFS byly umístěné pouze na jednom stejném stroji (viz kap. 6.1).

Pro analýzu a extrakci byly zvoleny extraktory „pe“ a „sqlite/*“, které jsou implementovány v původním systému Plaso i v nově implementoaném PySpark Plaso. Extrahované události byly v obou případech ukládány do JSON souboru, což znamená, že v případě PySpark Plaso nedošlo k nahrání událostí do distribuované databáze Halyard/HBase a v případě původního Plaso byl součástí experimentu i export událostí z interní databáze do zmiňovaného JSON souboru.

Původní plaso bylo spuštěno následujícími příkazy:

```
SQLITE_PARSERS=$(log2timeline.py --parsers list \
    | grep -o 'sqlite/[^ ]*' | tr '\n' ',')
log2timeline.py --parsers "pe,${SQLITE_PARSERS}" \
    results1.plaso test_data
psort.py -o json -w results1.json results1.plaso
```

[17] využívá se však více procesorů v jednom uzlu, což umí i PySpark Plaso
[18] https://sourceforge.net/projects/hdparm/
[19] https://github.com/log2timeline/plaso/tree/20200227/test_data

V případě nové implementace PySpark Plaso byla hlavní komponenta nasazeného systému volána příkazem:

```
wget "http://localhost/pyspark-plaso/api/extract" \
    -O results2.json
```

6.3 Výsledky experimentů

Spuštění původního Plaso s vybranými extraktory nad vstupními daty pro experimenty vygenerovalo 3018 událostí ve 26 automaticky vybraných souborech za 16 sekund (13 sekund extrakce a 3 sekundy ukládání do JSON souboru). Nová implementace PySpark Plaso za stejných podmínek vygenerovala 4185 událostí ve 32 souborech za 9 sekund. Tabulka 1 uvádí analyzované soubory a počty získaných událostí.

Ověření funkčnosti Z počtu extrahovaných událostí i z jejich seznamu v tabulce 1 je vidět, že v některých případech nebyly zvoleny stejné soubory pro extrakci. Zejména v případě nové implementace PySpark Plaso nebylo, oproti původní implementaci Plaso, zvoleno 5 souborů a nebylo takto nalezeno celkem 304 událostí. Tato skutečnost je dána odlišnou implementací fáze analýzy vstupních dat v obou projektech (z projektu Plaso byly převzaty pouze extraktory, nikoliv předchozí kód analýzy vstupních dat). Na druhou stranu bylo novou implementací PySpark Plaso vybráno, navíc oproti původní implementaci Plaso, 12 souborů a z nich získáno 864 událostí, které původní Plaso nenalezlo. Z hlediska funkčnosti lze tedy považovat obě implementace za rovnocenné.

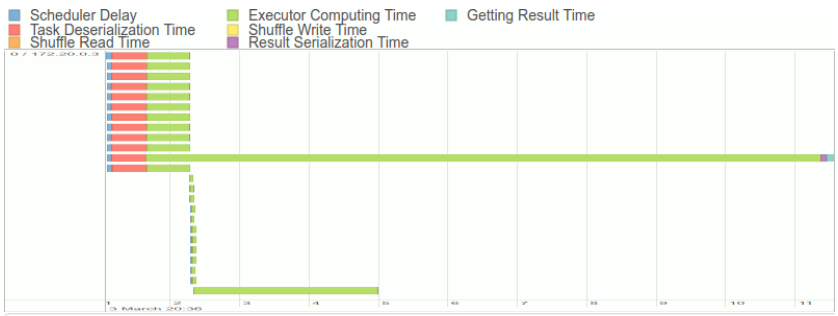
Ověření výkonu Z hlediska rychlosti, pomineme-li vliv rozdílných souborů automaticky vybraných k extrakci a vliv odlišného počtu extrahovaných událostí, je rychlejší nová implementace PySpark Plaso (9 sekund vs. 16 sekund) a to 46 procent. Při lepší škálovatelnosti nasazení PySpark Plaso, tj. distribuovaném uložení vstupních dat a distribuovaném výpočtu přes několik uzlů, lze očekávat výrazné urychlení celého procesu extrakce. Při výpočtu analýzy vstupů a extrakce událostí v PySpark Plaso na platformě Spark proběhlo celkem 24 Spark úloh zařazených v celkem jedné fázi (anglicky „stage“), což znamená, že v případě zapojení více výpočetních uzlů je možno celý proces řešit od začátku až po konec na stejných uzlech a nezdržovat se vzájemnou výměnou dat. Průběh zmiňovaných Spark úloh popisuje obrázek 6. Návaznosti jednotlivých fází a jejich kroků pak popisuje obrázek 7.

7 Závěr

Představené řešení navazuje na předchozí výzkum v oblasti extrakce událostí ze sociálních sítí uskutečněný v rámci projektu Tarzan v předchozích letech. Doplňuje již existující infrastrukturu o možnost extrakce událostí ze souborových

soubor	PySpark Plaso	původní Plaso
activity.sqlite	44	-
application_usage.sqlite	5	5
contacts2.db	5	5
cookies.db	1755	1755
Cookies-68.0.3440.106	-	16
document_versions.sql	-	4
downloads.sqlite	2	2
firefox_cookies.sqlite	295	295
googlehangouts.db	14	-
History	71	71
History.db	25	-
History-57.0.2987.133	2	2
History-58.0.3029.96	2	2
History-59_added-fake-column	2	2
History-59.0.3071.86	2	2
imessage_chat.db	10	10
kik_ios.sqlite	60	-
mackeeper_cache.db	198	198
mac_knowledgec-10.13.db	-	51
mac_knowledgec-10.14.db	-	229
mmssms.db	9	9
MyVideos107.db	4	-
places_new.sqlite	84	84
places.sqlite	202	202
quarantine.db	14	-
skype_main.db	24	24
snapshot.db	30	-
tango_android_profile.db	115	-
tango_android_tc.db	43	-
test_driver.sys	1	1
test_pe.exe	3	3
twitter_android.db	850	-
twitter_ios.db	184	-
Web DataNode	-	4
webviewCache.db	10	10
webview.db	8	8
windows_timeline_ActivitiesCache.db	112	-
celkem extrahovaných událostí	4185	3018
celkový čas extrakce a uložení	9 sekund	$13 + 3$ sekund

Tabulka 1. Automaticky vybrané soubory ze vstupních dat a počty z nich extrahovaných událostí.



Metric	Min	25th percentile	Median	75th percentile	Max
Duration	49 ms	53 ms	0.6 s	0.6 s	10 s
Scheduler Delay	7 ms	21 ms	62 ms	70 ms	88 ms
Task Deserialization Time	1 ms	3 ms	0.5 s	0.5 s	0.5 s
GC Time	0 ms	0 ms	17 ms	17 ms	17 ms
Result Serialization Time	0 ms	0 ms	0 ms	1 ms	81 ms

Obrázek 6. Průběh Spark úloh pro analýzu vstupních souborů a extrakci událostí z experimentálních dat v PySpark Palso.

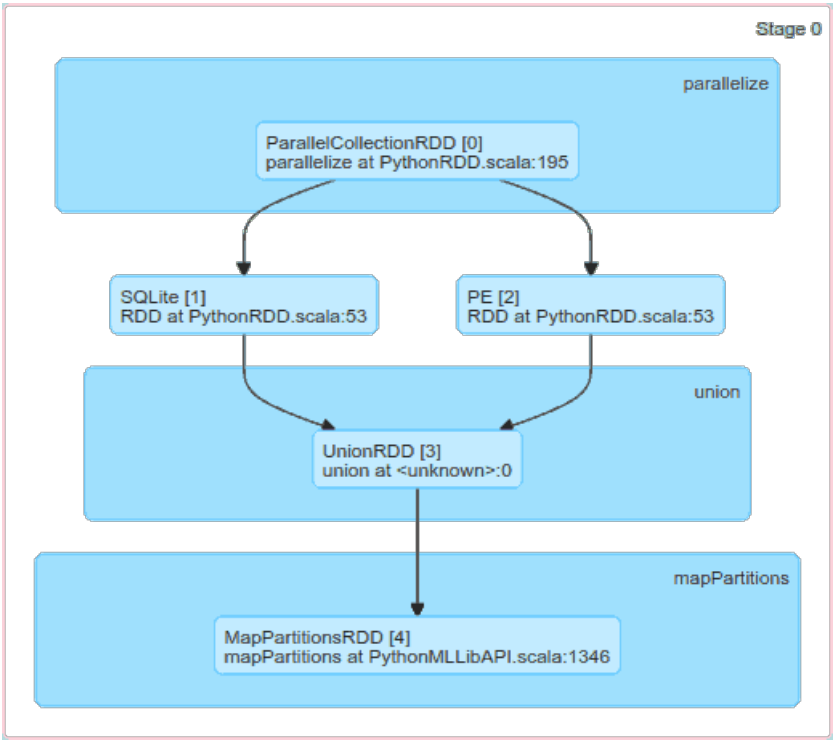
systémů a umožňuje integraci získaných dat do společné časové osy s daty ze sociálních sítí.

Vzhledem k poněkud exotické kombinaci platformy založené na jazycích Java a Scala a nástrojů vytvořených v jazyce Python se zvolené technické řešení ukázalo jako poměrně obtížné, všechny implementační problémy se však podařilo překonat. Systém byl implementován a jak vyplývá z výsledků testování uvedených v kapitole 6, nabízí funkčnost srovnatelnou s existujícími nástroji při dosažení vyšší efektivity díky paralelnímu spouštění jednotlivých extraktorů.

Z hlediska dalšího výzkumu a vývoje nabízí tato oblast několik zajímavých témat, na která by bylo možné se v budoucnu zaměřit. Jedná se například o rozšiřování množiny podporovaných extraktorů, zajímavou výzvou je pak rovněž automatická detekce vysokoúrovňových událostí například pomocí logických odvozovacích pravidel. Zvolený databázový model RDF použitý v našem řešení představuje z tohoto pohledu nadějný výchozí bod.

Reference

1. Burget, R.: Sociální sítě: Sběr a analýza dat v souvislosti s bezpečnostními incidenty. Technická Zpráva FIT-TR-2017-11, FIT VUT v Brně, 2017.
2. Burget, R.: Distribuované zpracování a analýza dat ze sociálních sítí: Návrh a implementace distribuované architektury. Technická Zpráva FIT-TR-2018-07, FIT VUT v Brně, 2018.
   URL https://www.fit.vut.cz/research/publication/11883



Obrázek 7. Spark transformace pro analýzu vstupních souborů a extrakci událostí v PySpark Palso a jejich zařazení do fází.

3. Chabot, Y.; Bertaux, A.; Nicolle, C.; aj.: An ontology-based approach for the reconstruction and analysis of digital incidents timelines. *Digital Investigation*, ročník 15, 2015: s. 83 – 100, ISSN 1742-2876, doi:https://doi.org/10.1016/j.diin.2015.07.005, special Issue: Big Data and Intelligent Data Analysis.
URL http://www.sciencedirect.com/science/article/pii/S1742287615000869
4. Guðjónsson, K.: Mastering the Super Timeline With log2timeline. SANS, Srpen 2010, [Online; navštíveno 23.11.2018].
URL https://www.sans.org/reading-room/whitepapers/logging/paper/33438
5. Han, X.; Wang, L.; Xu, S.; aj.: Linking social network accounts by modeling user spatiotemporal habits. In *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, 2017, s. 19–24.
6. Hargreaves, C.; Patterson, J.: An automated timeline reconstruction approach for digital forensic investigations. *Digital Investigation*, ročník 9, 2012: s. S69 – S79, ISSN 1742-2876, doi:https://doi.org/10.1016/j.diin.2012.05.006, the Proceedings of the Twelfth Annual DFRWS Conference.
URL http://www.sciencedirect.com/science/article/pii/S174228761200031X
7. Lanthaler, M.; Wood, D.; Cyganiak, R.: RDF 1.1 Concepts and Abstract Syntax. W3C recommendation, W3C, Únor 2014, http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.
8. Okolica, J. S.: *Temporal Event Abstraction and Reconstruction*. Dizertační práce, Air Force Institute of Technology Wright-Patterson AFB United States, Prosinec 2017.
9. Olsson, J.; Boldt, M.: Computer forensic timeline visualization tool. *Digital Investigation*, ročník 6, 2009: s. S78 – S87, ISSN 1742-2876, doi:https://doi.org/10.1016/j.diin.2009.06.008, the Proceedings of the Ninth Annual DFRWS Conference.
URL http://www.sciencedirect.com/science/article/pii/S1742287609000425
10. Ryšavý, O.; Rychlý, M.: Platforma pro zpracování dat síťové forenzní analýzy: Návrh a implementace prototypu. Technická Zpráva FIT-TR-2017-07, FIT VUT v Brně, 2017.
11. Schatz, B. L.; Mohay, G. M.; Clark, A.: Rich Event Representation for Computer Forensics. In *Asia Pacific Industrial Engineering and Management Systems APIEMS 2004*, 2004.
12. Schelkoph, D.; Peterson, G.; Okolica, J.: Digital Forensics Event Graph Reconstruction. In *10th EAI International Conference on Digital Forensics and Cyber Crime*, Sep 2018.
13. The Plaso Project Authors: Developer Guide – Plaso 20200430 documentation. Duben 2020, [Online; navštíveno 20.5.2020].
URL https://plaso.readthedocs.io/en/stable/sources/developer/Developers-Guide.html
14. White, T.: *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., první vydání, 2009, ISBN 0596521979, 9780596521974.